create database assignment\_1;

use assignment\_1;

create table city

(

Id int,

Name varchar(20),

CountryCode varchar(3),

District varchar(20),

Population int

);

insert into city values(6,'Rotterdam','NLD','Zuid-Halland',593321);

insert into CITY VALUES(3878,'Scottsdale','USA','Arizona',202705);

insert into CITY VALUES(3965,'Corona','USA','California',124966);

insert into CITY VALUES(3973,'Concord','USA','California',121780);

insert into CITY VALUES(3977,'Cedar Rapids','USA','Iowa',120758);

insert into CITY VALUES(3982,'Coral Springs','USA','Florida',117549);

insert into CITY VALUES(4054,'Fairfield','USA','California',92256);

insert into CITY VALUES(4058,'Boulder','USA','Colorado',91238);

insert into CITY VALUES(4061,'Fall River','USA','Massachusetts',90555);

select \* from city;

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

­­---> select \* from city where countryCode='USA' and Population > 100000;

Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.

-🡪 select name from city where countryCode='USA' and Population > 120000;

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

* select \* from city;

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

🡪select \* from city where id=1661;

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

🡪select \* from city where countryCode='JPN';

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

* select name from city where countryCode='JPN';

use assignment\_1;

create table Station

(

id int,

city varchar(21),

state varchar(2),

Lat\_N int,

LONG\_W INT

);

describe station;

INSERT INTO STATION VALUES(794,'Kissee Mills','MO',139,73);

INSERT INTO STATION VALUES(824,'Loma Mar','CA',48,130);

INSERT INTO STATION VALUES(603,'Sandy Hook','CT',72,148);

INSERT INTO STATION VALUES(478,'Tipton','IN',33,97);

INSERT INTO STATION VALUES(619,'Arlington','CO',75,92);

INSERT INTO STATION VALUES(711,'Turner','AR',50,101);

INSERT INTO STATION VALUES(839,'Slidell','LA',85,151);

INSERT INTO STATION VALUES(411,'Negreet','LA',98,105);

INSERT INTO STATION VALUES(588,'Glencoe','KY',46,136);

INSERT INTO STATION VALUES(665,'Chelsea','IA',98,59);

INSERT INTO STATION VALUES(342,'Chignik Lagoon','AK',103,153);

INSERT INTO STATION VALUES(733,'Pelahatchie','MS',38,28);

INSERT INTO STATION VALUES(811,'Dorrance','KS',102,121);

INSERT INTO STATION VALUES(698,'Albany','CA',49,80);

INSERT INTO STATION VALUES(325,'Monument','KS',70,141);

INSERT INTO STATION VALUES(414,'Manchester','MD',73,37);

INSERT INTO STATION VALUES(113,'Prescott','IA',39,65);

INSERT INTO STATION VALUES(971,'Graettinger','IA',94,150);

INSERT INTO STATION VALUES(266,'Cahone','CO',116,127);

select \* from station;

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

-🡪 select city,state from station;

Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results

in any order, but exclude duplicates from the answer.

* select distinct(city) from station where id %2=0 order by city asc;

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

distinct CITY entries in the table.

* select count(city) as total\_number\_of\_city, count(distinct(city)) as total\_Number\_of\_distinct\_city,

(count(city)- count(distinct(city))) as difference\_city\_count 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.

For example, CITY has four entries: DEF, ABC, PQRS and WXY.

Sample Output ABC 3 PQRS 4

Hint - When ordered alphabetically, the CITY names are listed as ABC, DEF, PQRS, and WXY, with lengths and. The longest name is PQRS, but there are options for shortest named city. Choose ABC, because it comes first alphabetically. Note You can write two separate queries to get the desired output. It need not be a single query.

-🡪

select city,length(city) as min\_length\_of\_city from station order by length(city) asc limit 1;

select city,length(city)as max\_length\_of\_city from station order by length(city) desc 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.

🡪

1] select distinct city from station where

lcase(city) Like 'a%' or

lcase(city) Like 'e%' or

lcase(city) Like 'i%' or

lcase(city) Like 'o%' or

lcase(city) Like 'u%' order by city;

2] SELECT DISTINCT(CITY) AS DISTINCT\_CITY\_NAME FROM STATION WHERE

lower(SUBSTR(city,1,1)) in ('o','e','i','a','u');

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

🡪

1] select distinct(city) from station where lower(substr(city,-1,1)) in ('a','e','i','o','u') order by city;

2] select distinct city from station where

lcase(city) Like '%a' or

lcase(city) Like '%e' or

lcase(city) Like '%i' or

lcase(city) Like '%o' or

lcase(city) Like '%u'

order by city;

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

🡪

select distinct city from station where

lcase(city) not like 'a%' OR

lcase(city) not like 'e%' OR

lcase(city) not like 'i%' OR

lcase(city) not like 'o%' OR

lcase(city) not like 'u%'

order by city;;

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

🡪

select distinct city from station where

lcase(city) not like '%a' OR

lcase(city) not like '%e' OR

lcase(city) not like '%i' OR

lcase(city) not like '%o' OR

lcase(city) not like '%u'

order by city;

Q15. Query the list of CITY names from STATION that either do not start with vowels or do not end

with vowels. Your result cannot contain duplicates.

* select distinct city from station where city not regexp '^[aeiou]' or city not regexp '[aeiou]$';

Q16. Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

* select distinct city from station where city not regexp '^[aeiou]' and city not regexp '[aeiou]$';

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

create table if not exists product

(

product\_id int not null,

product\_name varchar(50),

unit\_price i

constraint pk primary key (product\_id)

);

show create table product;

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

select \* from product;

delete from product where product\_id is null;

Create table if not exists Sales

(

seller\_id int,

product\_id int,

buyer\_id int,

sale\_date date,

quantity int,

price int,

constraint fk foreign key (product\_id) references product(product\_id)

);

insert into sales values(1,1,1,'2019-01-21',2,2000);

insert into sales values(1,2,2,'2019-02-17',1,800);

insert into sales values(2,2,3,'2019-06-02',1,800);

insert into sales values(3,3,4,'2019-05-13',2,2800);

select \* from sales;

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

* select \* from product where product\_id not in

(select product\_id from sales where sale\_date

not between "2019-01-01" and "2019-03-31");

create table if not exists views

(

article\_id int,

author\_id int,

viewer\_id int,

view\_date date

);

insert into views values(1,3,5,'2018-08-01'),

(1,3,6,'2019-08-02'),(2,7,7,'2019-08-01'),

(2,7,6,'2019-08-02'),(4,7,1,'2019-07-22'),

(2,4,4,'2019-07,21'),(3,4,4,'2019-07-21');

select \* from views;

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

* select distinct author\_id as id

from views where author\_id=viewer\_id

order by id asc;

Create table if not exists Delivery

(

delivery\_id int,

customer\_id int,

order\_date date,

customer\_pref\_delivery\_date date,

constraint pk primary key(delivery\_id)

);

--- drop table delivery;

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');

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

🡪

1]select round(100\*d2.immediate\_orders /count(d1.delivery\_id),2)

as immediate\_percentage from delivery d1,(select count(order\_date) as immediate\_orders

from delivery where (order\_date=customer\_pref\_delivery\_date)) d2;

2] select round(sum(if(order\_date=customer\_pref\_delivery\_date,1,0))/count(\*)\*100,2)

as immediate\_percentage from delivery;

use assignment\_1;

create table if not exists Ads

(

ad\_id int,

user\_id int,

action enum('Clicked','Viewed','Ignored'),

constraint pk primary key(ad\_id,user\_id)

);

insert into ads values(1,1,'Clicked'),

(2,2,'Clicked'),(3,3,'Viewed'),

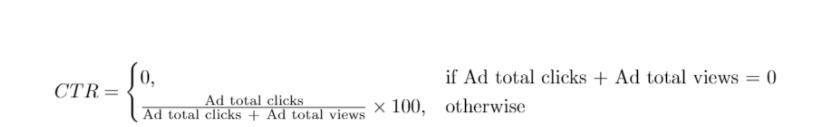
(5,5,'Ignored'),(1,7,'Ignored'),

(2,7,'Viewed'),(3,5,'Clicked'),(1,4,'Viewed'),

(2,11,'Viewed'),(1,2,'Clicked');

select \* from Ads;

Q.20 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.



Explanation:

for ad\_id = 1, ctr = (2/(2+1)) \* 100 = 66.67

for ad\_id = 2, ctr = (1/(1+2)) \* 100 = 33.33

for ad\_id = 3, ctr = (1/(1+1)) \* 100 = 50.00

for ad\_id = 5, ctr = 0.00, Note that ad\_id = 5 has no clicks or views. Note that we do not care about Ignored Ads

🡪

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 Ads

group by ad\_id

order by ctr desc,ad\_id asc;

create table if not exists employee

(

employee\_id int,

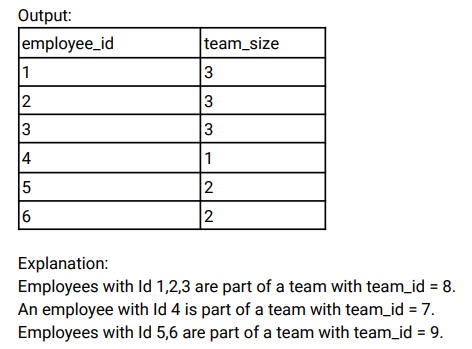
team\_id int,

constraint pk primary key(employee\_id)

);

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

Q.21 Write an SQL query to find the team size of each of the employees. Return result table in any order.



* select employee\_id,count(team\_id) over(partition by team\_id) as team\_size

from employee order by employee\_id;

create table if not exists countries

(

country\_id int,

country\_name varchar(50),

constraint pk primary key(country\_id)

);

insert into countries values(2,'USA'),(3,'Australia'),(7,'Peru'),

(5,'China'),(8,'Morocco'),(9,'Spain');

select \* from countries;

create table weather

(

country\_id int,

weather\_state int,

day date,

constraint pk primary key(country\_id,day)

);

desc weather;

insert into weather values(2,15,'2019-11-01'),

(2,12,'2019-10-28'),(2,12,'2019-10-27'),(3,-2,'2019-11-10'),

(3,0,'2019-11-11'),(3,3,'2019-11-12'),(5,16,'2019-11-07'),(5,18,'2019-11-09'),

(5,21,'2019-11-23'),(7,25,'2019-11-28'),(7,22,'2019-12-01'),

(7,20,'2019-12-02'),(8,25,'2019-11-05'),(8,27,'2019-11-15'),(8,31,'2019-11-25'),

(9,7,'2019-10-23'),(9,3,'2019-12-23');

select \* from weather;

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

--- The type of weather is:

--- ● Cold if the average weather\_state is less than or equal 15,

--- ● Hot if the average weather\_state is greater than or equal to 25, and

--- ● Warm otherwise. Return result table in any order

* select c.country\_name,

case

when avg(w.weather\_state) <=16 then 'Cold'

when avg(w.weather\_state) >=25 then 'Hot'

else 'Warm'

end as weather\_type

from countries c

inner join weather w on w.country\_id=c.country\_id

where day between '2019-11-01' and '2019-11-30'

group by c.country\_id;

use assignment\_1;

create table if not exists prices

(

product\_id int,

start\_date date,

end\_date date,

price int,

constraint pk primary key(product\_id,start\_date,end\_date)

);

insert into Prices VALUES

(1,'2019-02-17','2019-02-28',5),(1,'2019-03-01','2019-03-22',20),

(2,'2019-02-01','2019-02-20',15),(2,'2019-02-21','2019-03-31',30);

select \* from Prices;

create table if not exists unitsold

(

product\_id int,

purchase\_date date,

units int

);

insert into UnitSold VALUES

(1,'2019-02-25',100),(1,'2019-03-01',15),

(2,'2019-02-10',200),(2,'2019-03-22',30);

select \* from UnitSold;

**Q.23** Write an SQL query to find the average selling price for each product. average\_price should be

--- rounded to 2 decimal places

select u.product\_id,round(sum(u.units\*p.price)/sum(u.units),2) as average\_price

from unitsold u join prices p

on(u.product\_id=p.product\_id and

u.purchase\_date >= p.start\_date and

u.purchase\_date <= p.end\_date) group by product\_id;

create table if not exists Activity

(

player\_id int,

device\_id int,

event\_date date,

games\_played INT DEFAULT 0,

constraint pk PRIMARY KEY (player\_id, event\_date)

);

INSERT into Activity values

(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),(3,4,'2018-07-03',5);

select \* from activity;

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

--- player.Return the result table in any order.

1]

select player\_id,first\_value(event\_date) over(partition by player\_id ) first\_login

from activity group by player\_id;

2]

select

tmp.player\_id,tmp.event\_date as first\_login

from (select \*,

row\_number() over(partition by player\_id ) as row\_num

from Activity) tmp

where tmp.row\_num = 1;

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

* select player\_id,first\_value(device\_id) over(partition by player\_id ) device\_id

from activity group by player\_id;

* select tmp.player\_id,tmp.device\_id from

(select \*, row\_number() over(partition by player\_id) as row\_num from activity) tmp where tmp.row\_num=1;

create table if not exists products

(

product\_id int,

product\_name varchar(40),

product\_category varchar(40),

constraint pk primary key(product\_id)

);

insert into products values(1,'Leetcode Solutions','Book'),(2,'Jewels of Stringology','Book'),

(3,'HP','Laptop'),(4,'Lenovo','Laptop'),(5,'Leetcode Kit','T-shirt');

select \* from Products;

create table if not exists Orders

(

product\_id int,

order\_date date,

unit int,

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

insert into Orders values

(1,'2020-02-05',60),(1,'2020-02-10',70),(2,'2020-01-18',30),(2,'2020-02-11',80),

(3,'2020-02-17',2),(3,'2020-02-24',3),(4,'2020-03-01',20),(4,'2020-03-04',30),

(4,'2020-03-04',60),(5,'2020-02-25',50),(5,'2020-02-27',50),(5,'2020-03-01',50);

select \* from Orders;

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

**-->** select p.product\_name,sum(o.unit) as unit from products p join orders o

on p.product\_id=o.product\_id where o.order\_date between '2020-02-01' and '2020-02-28'

group by p.product\_id having sum(unit)>=100 ;

create table if not exists Users

(

user\_id int,

name varchar(50),

mail varchar(50),

constraint pk PRIMARY KEY (user\_id)

);

insert into Users VALUES

(1,'Winston','winston@leetcode.com'),(2,'Jonathan','jonathanisgreat'),

(3,'Annabelle','bella-@leetcode.com'),(4,'Sally','sally.come@leetcode.com'),

(5,'Marwan','quarz#2020@leetcode.com'),

(6,'David','david69@gmail.com'),(7,'Shapiro','.shapo@leetcode.com');

select \* from Users;

**Q.27** Write an SQL query to find the users who have valid emails. A valid e-mail has a prefix name and a domain where:

● The prefix name is a string that may contain letters (upper or lower case), digits, underscore '\_', period '.', and/or dash '-'. The prefix name must start with a letter.

● The domain is '@leetcode.com'.

* select \* from users where REGEXP\_LIKE(mail,'^[A-Za-z][a-zA-z0-9\\_\.\-][\*@leetcode.com](mailto:*@leetcode.com)');

use assignment\_1;

create table if not exists Customers

(

customer\_id int,

name varchar(50),

country varchar(50),

constraint pk PRIMARY KEY (customer\_id)

);

insert into Customers VALUES

(1,'Winston','USA'),(2,'Jonathan','Peru'),(3,'Moustafa','Egypt');

select \* from Customers;

--- Table: Product

create table if not exists Product1

(

product\_id int,

description varchar(255),

price int,

constraint pk PRIMARY KEY (product\_id)

);

insert into Product1 values (10,'LC Phone',300),(20,'LC

T-Shirt',10),(30,'LC Book',45),(40,'LC Keychain',2);

select \* from Product1;

--- Table: Orders

create table if not exists order1

(

order\_id int,

customer\_id int,

product\_id int,

order\_date DATE,

quantity int,

constraint pk PRIMARY KEY(order\_id)

);

insert into Order1 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 \* from Order1;

**Q.28** Write an SQL query to report the customer\_id and customer\_name of customers who have spent at least $100 in each month of June and July 202

* select c.customer\_id,c.name from customers c, order1 o,product1 p

where c.customer\_id=o.customer\_id and p.product\_id=o.product\_id

group by o.customer\_id

having

(

sum(case when o.order\_date like '2020-06%' then o.quantity\*p.price else 0 end) >=100

and

sum(case when o.order\_date like '2020-07%' then o.quantity\*p.price

else 0 end) >= 100

);

use assignment\_1;

create table if not exists TVProgram

(

program\_date date,

content\_id int,

channel varchar(20),

constraint pk primary key(program\_date,content\_id)

);

show create table TVProgram;

--- drop table tvprogram;

insert into TVProgram VALUES ('2020-06-10 08:00',1,'LC-Channel'),('2020-05-11 12:00',2,'LC-Channel'),

('2020-05-12 12:00',3,'LC-Channel'),('2020-05-13 14:00',4,'Disney Ch'),

('2020-06-18 14:00',4,'Disney Ch'),('2020-07-15 16:00',5,'Disney Ch');

select \* from TVProgram;

create table if not exists content

(

content\_id int,

title varchar(40),

kids\_content enum('Y','N'),

content\_type varchar(40),

constraint pk primary key(content\_id)

);

insert into Content VALUES (1,'Leetcode Movie','N','Movies'),

(2,'Alg.for Kids','Y','Series'),(3,'DatabaseSols','N','Series'),

(4,'Aladdin','Y','Movies'),(5,'Cinderella','Y','Movies');

select \* from Content;

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

🡪 select c.title from content c inner join TVProgram t

on t.content\_id=c.content\_id where c.kids\_content='Y' and

content\_type = 'Movies' and

t.program\_date like '2020-06%' ;

create table if not exists NPV

(

id int,

year int,

npv int,

constraint pk 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);

select \* from NPV;

--- Table: Queries

create table if not exists Queries

(

id int,

year int,

constraint pk PRIMARY KEY (id, year)

);

insert into Queries VALUES (1,2019),(2,2008),(3,2009),

(7,2018),(7,2019),(7,2020),(13,2019);

select \* from Queries;

**V. Impt** **Q.30** Write an SQL query to find the npv of each query of the Queries table.

Coalesce function is used to handle the Null values. The null values are replaced with user- defined values during the expression evaluation process.

🡪 select q.id,q.year,Coalesce(n.npv,0) from queries q left join npv n on

q.id=n.id and q.year=n.year;

create table if not exists Employees

(

id int,

name varchar(20),

constraint pk primary key(id)

);

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

select \* from Employees;

--- Table: EmployeeUNI

create table if not exists EmployeeUNI

(

id int,

unique\_id int,

constraint pk PRIMARY KEY (id, unique\_id)

);

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

select \* from EmployeeUNI;

Q.32 Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

**🡪**  select en.unique\_id,e.name from employeeUNI en right join

employees e on if(e.id=en.id,e.id,null);

* select e.unique\_id,e1.name from employees e1 left join employeeuni e

on e1.id=e.id group by e1.id order by e1.name asc;

create table if not exists users1

(

id int,

name varchar(20),

constraint pk primary key(id)

);

insert into Users1 VALUES

(1,'Alice'),(2,'Bob'),(3,'Alex'),

(4,'Donald'),(7,'Lee'),(13,'Jonathan'),(19,'Elvis');

select \* from users1;

create table if not exists Rides

(

id int,

user\_id int,

distance int,

constraint pk primary key(id),

constraint fk1 foreign key(user\_id) references users1(id)

);

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 \* from Rides;

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

--- Return the result table ordered by travelled\_distance in descending order, if two or more users

--- travelled the same distance, order them by their name in ascending order

select distinct(u.name),sum(ifnull(r.distance, 0)) over(partition by u.name )

as travelled\_distance from rides r right join users1 u

on r.user\_id=u.id order by r.distance desc,u.name asc;

select u.name,sum(ifnull(r.distance, 0))

as travelled\_distance from rides r right join users1 u

on r.user\_id=u.id group by u.name order by 2 desc,1 asc;

Table: Products

create table if not exists Products

( product\_id int, product\_name varchar(50), product\_category VARCHAR(50), constraint pk PRIMARY KEY (product\_id) );

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

select \* from Products;

--Table: Orders

create table if not exists Orders

( product\_id int, order\_date date, unit int );

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);

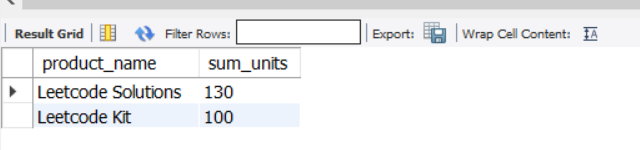
select \* from Orders;

**Q.34**Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

🡪 select p.product\_name,sum(o.unit) as sum\_units from products p right join orders o

on p.product\_id=o.product\_id where o.order\_date

between '2020-02-01' and '2020-02-28' group by o.product\_id having sum(o.unit)>=100;



create table if not exists movies

(

movie\_id int,

title varchar(20),

constraint pk primary key(movie\_id)

);

insert into movies values(1,'Avengers'),(2,'Fozen 2'),(3,'Joker');

select \* from movies;

create table if not exists users2

(

user\_id int,

name varchar(20),

constraint pk primary key(user\_id)

);

insert into users2 values(1,'Daniel'),(2,'Monica'),(3,'Maria'),(4,'James');

select \* from users2;

create table if not exists MovieRating

(

movie\_id int,

user\_id int,

rating int,

created\_at date,

constraint pk PRIMARY KEY (movie\_id, user\_id)

);

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 \* from MovieRating;

**Q.35** Write an SQL query to:

--- ● Find the name of the user who has rated the greatest number of movies. In case of a tie,

--- return the lexicographically smaller user name.

--- ● Find the movie name with the highest average rating in February 2020. In case of a tie, return

--- the lexicographically smaller movie name

🡪

1]

select user\_name as results from

(

select u.name as user\_name,count(\*) as counts

from users2 u

join MovieRating m on m.user\_id=u.user\_id group by m.user\_id

order by counts desc,counts asc limit 1) first\_query

union

select movie\_name as results from

(

select m1.title as movie\_name,avg(m.rating) as rate from MovieRating m

join movies m1 on m1.movie\_id = m.movie\_id

WHERE substr(m.created\_at, 1, 7) = '2020-02'

GROUP BY m.movie\_id

ORDER BY rate DESC, movie\_name ASC LIMIT 1

) second\_query;

2]

select user\_name as results from (

select u.name as user\_name, count(\*) as rate from movierating m

join users2 u on u.user\_id=m.user\_id group by user\_name order by 2 desc,1 asc limit 1

)first\_query

Union

select movie\_name as results from (

select m1.title as movie\_name,avg(m.rating) as rating from movierating m

join movies m1 on m1.movie\_id=m.movie\_id where m.created\_at like '2020-02%'

group by movie\_name order by 2 desc,1 asc limit 1

)second\_query;

create table if not exists Users3

(

id int,

name varchar(50),

constraint pk PRIMARY KEY (id)

);

insert into Users3 VALUES

(1,'Alice'),(2,'Bob'),(3,'Alex'),(4,'Donald'),(7,'Lee'),(13,'Jonathan')

,(19,'Elvis');

select \* from Users3;

--- Table: Rides

create table if not exists Rides1

(

id int,

user\_id int,

distance int,

constraint pk PRIMARY KEY (id),

constraint fk4 FOREIGN KEY (user\_id) REFERENCES Users3(id)

);

insert into Rides1 VALUES

(1,1,120),(2,2,317),(3,3,222),(4,7,100),(5,13,312),(6,19,50),(7,7,120),

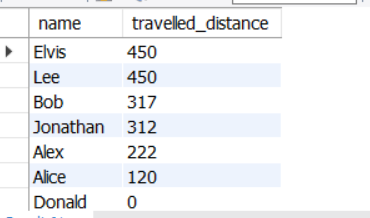
(8,19,400),(9,7,230);

select \* from Rides1;

**Q.36** Write an SQL query to report the distance travelled by each user.

* Return the result table ordered by travelled\_distance in descending order, if two or more users
* travelled the same distance, order them by their name in ascending order.
* select u.name,coalesce(sum(r.distance),0) as travelled\_distance from rides1 r

right join users3 u on u.id=r.user\_id group by u.name order by travelled\_distance desc,u.name asc;



create table if not exists Departments

(

id int,

name varchar(40),

constraint pk primary key(id)

);

insert into departments values(1,'Electrical Engineering'),

(7,'Computer Engineering'),(13,'Business Administration');

select \* from departments;

create table if not exists students

(

id int,

name varchar(40),

department\_id int,

constraint pk primary key(id)

);

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 \* from Students;

**Q.38** Write an SQL query to find the id and the name of all students who are enrolled

--- in departments that no longer exist.

select s.id,s.name from students s

left join departments d on d.id=s.department\_id where d.id is null ;

create table if not exists 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 Calls;

**Q.39** Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2.

🡪

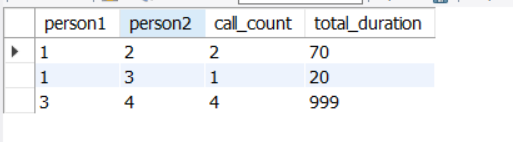
select least(from\_id,to\_id) as person1,

greatest(from\_id,to\_id) as person2,

count(\*) as call\_count,

sum(duration) as total\_duration

from calls group by 1,2;



create table if not exists Warehouse

(

name varchar(20),

product\_id int,

units int,

constraint pk primary key(name,product\_id)

);

insert into Warehouse values('LCHouse1',1,1),('LCHouse1',2,10),('LCHouse1',3,5),

('LCHouse2',1,2),('LCHouse2',2,2),('LCHouse3',4,1);

select \* from Warehouse;

create table if not exists products1

(

product\_id int,

product\_name varchar(30),

width int,length int,

height int,

constraint pk primary key(product\_id)

);

insert into Products1 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 \* from Products1;

**Q.41** Write an SQL query to report the number of cubic feet of volume the

--- inventory occupies in each warehouse.

select w.name,sum(p.width\*p.length\*p.height\*w.units) as Volume from Warehouse w

join products1 p on p.product\_id=w.product\_id group by w.name;

* select name as warehouse\_name, sum(units \* vol) as volume from Warehouse w join (select product\_id, Width\*Length\*Height as vol from Products) p on w.product\_id = p.product\_id group by name;

create table if not exists sales1

(

sale\_date date,

fruit enum('apples','oranges'),

sold\_num int,

constraint pk primary key(sale\_date,fruit)

);

insert into Sales1 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 \* from sales1;

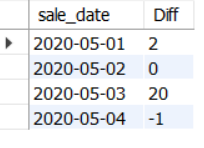
**Q.42** Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale\_date.

select a.sale\_date,(a.sold\_num - b.sold\_num) as Diff from sales1 a

left join sales1 b

on a.sale\_date=b.sale\_date

where a.fruit='apples' and b.fruit='oranges';



create table if not exists Activity1

(

player\_id int,

device\_id int,

event\_date date,

games\_played int,

constraint pk 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 \* from activity1;

**Q.43** Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

WITH CTE AS (

select player\_id,min(event\_date)as event\_start\_date

from activity1

group by player\_id

)

select

round((count(distinct c.player\_id)) / (select count(distinct player\_id)

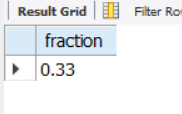
from activity1),2) as fraction

from CTE c join

activity1 a

on c.player\_id=a.player\_id and

datediff(c.event\_start\_date,a.event\_date)=-1;



create table if not exists employee1

(

id int,

name varchar(20),

department varchar(5),

manager\_id int,

constraint pk primary key(id)

);

insert into employee1 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 \* from Employee1;

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

* select a.name from employee1 a join employee1 b

on a.id=b.manager\_id group by a.name having count(distinct b.id) >= 5;

* select Name from Employee1

where Id in

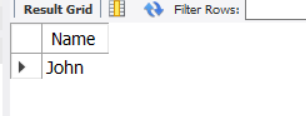
(

select Manager\_Id from Employee1

group by 1

having count(\*) >= 5

);



create table if not exists department

(

dept\_id int,

dept\_name varchar(20),

constraint pk primary key(dept\_id)

);

insert into Department VALUES

(1,'Engineering'),(2,'Science'),(3,'Law');

select \* from department;

create table if not exists student

(

student\_id int,

student\_name varchar(20),

gender varchar(2),

dept\_id int,

constraint pk primary key(student\_id),

constraint fk5 foreign key(dept\_id) references department(dept\_id)

);

insert into Student VALUES

(1,'Jack','M',1),(2,'Jane','F',1),(3,'Mark','M',2);

select \* from student;

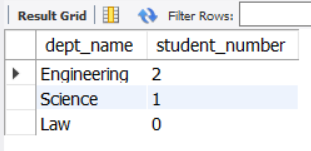
**Q.45** Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students). Return the result table ordered by student\_number in descending order. In case of a tie, order them by dept\_name alphabetically.



select d.dept\_name,coalesce(count(s.student\_id),0) as student\_number from department d

left join student s

on d.dept\_id=s.dept\_id group by d.dept\_id order by 2 desc,1 asc;



create table if not exists product2

(

product\_key int,

constraint pk primary key(product\_key)

);

insert into product2 values(5),(6);

create table if not exists customer

(

customer\_id int,

product\_key int,

constraint fk6 foreign key(product\_key) references product2(product\_key)

);

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

select \* from product2;

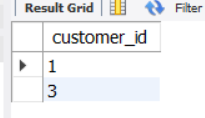
select \* from customer;

**Q.46** Write an SQL query to report the customer ids from the Customer table that bought

--- all the products in the Product table.

select c.customer\_id from customer c group by c.customer\_id

having count(distinct c.product\_key)= (select count(distinct product\_key) from product2);



create table if not exists employee2

(

employee\_id int,

name varchar(40),

experince\_year int,

constraint pk primary key(employee\_id)

);

insert into employee2 values(1,'Khaled',3),(2,'Ali',2),(3,'John',3),(4,'Doe',2);

select \* from Employee2;

create table if not exists project

(

project\_id int,

employee\_id int,

constraint pk primary key(project\_id,employee\_id),

constraint fk7 foreign key(employee\_id) references employee2(employee\_id)

);

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

**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.

select project\_id,employee\_id from

(

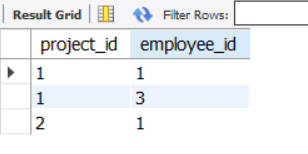
select p.project\_id,p.employee\_id,

DENSE\_RANK() over(partition by p.project\_id order by e.experince\_year desc) as rnk

from project p join employee2 e on p.employee\_id=e.employee\_id

)x

where rnk=1;



create table if not exists Books

(

book\_id int,

name VARCHAR(50),

available\_from date,

constraint pk PRIMARY KEY(book\_id)

);

insert into Books VALUES (1,'"Kalila And Demna"','2010-01-01'),

(2,'"28Letters"','2012-05-12'),(3,'"The Hobbit"','2019-06-10'),

(4,'"13 ReasonsWhy"','2010-01-01'),(5,'"The Hunger Games"','2008-09-21');

select \* from Books;

--- Table: Orders

create table if not exists Orders1

(

order\_id int,

book\_id int,

quantity int,

dispatch\_date date,

constraint pk PRIMARY KEY (order\_id),

constraint fk8 FOREIGN KEY (book\_id) REFERENCES Books(book\_id)

);

insert into Orders1 VALUES

(1,1,2,'2018-07-26'),(2,1,1,'2018-11-05'),(3,3,8,'2019-06-11'),

(4,4,6,'2019-06-05'),(5,4,5,'2019-06-20'),(6,5,9,'2009-02-02'),(7,5,8,'2010-04-13');

select \* from Orders1;

**Q.48** 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.

select Books.book\_id, name from Books join Orders1

on Books.book\_id = Orders1.book\_id

where available\_from < '2019-05-23'

and dispatch\_date between '2018-06-23' and '2019-06-23'

group by Books.book\_id

having sum(quantity) < 10

union

select book\_id,name from books

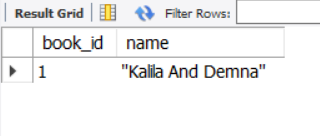
where available\_from < 2019-06-2023

and book\_id not in (

select distinct book\_id from orders1 where dispatch\_date between

'2018-06-23' and '2019-06-23'

);



create table if not exists Enrollments

(

student\_id int,

course\_id int,

grade int,

constraint pk 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 \* from Enrollments;

**Q.49** Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course\_id.

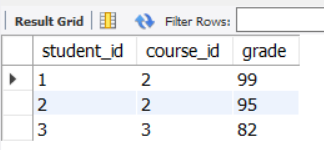
Return the result table ordered by student\_id in ascending order.

select e.student\_id,e.course\_id,e.grade from

( select \*,

row\_number() over(partition by student\_id order by grade desc) rn from enrollments) e

where e.rn=1;



create table if not exists Player

(

Player\_id int,

group\_id int,

constraint pk primary key(player\_id)

);

insert into player values(15,1),(25,1),(30,1),(45,1),

(10,2),(35,2),(50,2),(20,3),(40,3);

select \* from player;

create table if not exists Matches

(

match\_id int,

first\_player int,

second\_player int,

first\_score int,

second\_score int,

constraint pk PRIMARY KEY (match\_id)

);

insert into Matches VALUES

(1,15,45,3,0),(2,30,25,1,2),(3,30,15,2,0),(4,40,20,5,2),(5,35,50,1,1);

select \* from Matches;

**Q.50 Write an SQL query to find the winner in each group.**

🡪

select group\_id,player\_id from

(select group\_id,player\_id,sum((

case when p.player\_id=m.first\_player then m.first\_score

when p.player\_id=m.second\_player then m.second\_score

end

)) as totalscore

from player p,matches m where p.player\_id=m.first\_player or p.player\_id=m.second\_player

group by group\_id,player\_id

order by group\_id,totalscore desc,player\_id

) as temp

group by group\_id

order by group\_id,player\_id ;

