-- Q.1 Query all columns for all American cities in the CITY table with

-- populations larger than 100000. The countryCode for America is USA.

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),

(1661,'NYC','USA','Newyork',90525),

(4333,'Mitaka','JPN','Tokyo',91155),

(4551,'Hino','JPN','Tokyo',45678);

SELECT

id,

name,

countrycode,

district,

population

FROM

city

WHERE

population > 100000

AND

countrycode = 'USA';

-- Q.2 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

population > 120000

AND

countrycode = 'USA';

-- Q.3 Query all columns (attributes) for every row in the CITY table.

SELECT

id,

name,

countrycode,

district,

population

FROM

city;

-- Q.4 Query all columns for a city in CITY with the id 1661.

SELECT

id,

name,

countrycode,

district,

population

FROM

city

WHERE

id = 1661;

-- Q.5 Query all attributes of every Japanese city in the CITY table.

-- The countryCODE for Japan is JPN.

SELECT

id,

name,

countrycode,

district,

population

FROM

city

WHERE

countrycode = 'JPN';

-- Q.6 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';

-- Q.7 Query a list of CITY and STATE from the station table.

CREATE TABLE station(

id INT,

city VARCHAR(21),

state VARCHAR(2),

lat\_n INT,

long\_w INT

);

INSERT INTO station VALUES

(794,'Kissee Mills','MO',139,73),

(824,'Loma Mar','CA',48,130),

(603,'Sandy Hook','CT',72,148),

(478,'Tipton','IN',33,97),

(619,'Arlington','CO',75,92),

(711,'Turner','AR',50,101),

(839,'Slidell','LA',85,151),

(411,'Negreet','LA',98,105),

(588,'Glencoe','KY',46,136),

(665,'Chelsea','IA',98,59),

(342,'Chignik Lagoon','AK',103,153),

(733,'Pelahatchie','MS',38,28),

(441,'Hanna,City','IL',50,136),

(811,'Dorrance','KS',102,121),

(698,'Albany','CA',49,80),

(325,'Monument','KS',70,141),

(414,'Manchester','MD',73,37),

(113,'Prescott','IA',39,65),

(971,'Graettinger','IA',94,150),

(266,'Cahone','CO',116,127);

SELECT

city,

state

FROM

station;

-- Q.8 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;

-- Q.9 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) - COUNT(DISTINCT city)) AS difference

FROM

station;

-- Q.10 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.

-- APPROACH 1:

(SELECT

city,

Length(city) AS len

FROM

station

ORDER BY

Length(city),

city

LIMIT 1)

UNION ALL

(SELECT

city,

Length(city) AS len

FROM

station

ORDER BY

Length(city) DESC,

city

LIMIT 1);

-- APPROACH 2:

WITH temp\_city AS (

SELECT

city,

LENGTH(city) as len,

ROW\_NUMBER() OVER (ORDER BY LENGTH(city), city) AS smallest,

ROW\_NUMBER() OVER (ORDER BY LENGTH(city) DESC, city) AS largest

FROM

station

)

SELECT

city,

len

FROM

temp\_city

WHERE

smallest = 1

OR

largest = 1

ORDER BY

len;

-- 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.

-- Approach 1

SELECT

DISTINCT city

FROM

station

WHERE

LEFT(city , 1) IN ('a','e','i','o','u');

-- Approach 2

SELECT

DISTINCT city

FROM

station

WHERE

SUBSTR(city,1,1) IN ('A','E','I','O','U');

-- Approach 3

SELECT

DISTINCT city

FROM

station

WHERE

city RLIKE '^[aeiouAEIOU]';

-- Approach 4

SELECT

DISTINCT city

FROM

station

WHERE

city REGEXP '^[aeiouAEIOU]';

-- Approach 5

SELECT

DISTINCT city

FROM

station

WHERE

city REGEXP '^[aeiou]';

-- Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from station.

-- Your result cannot contain duplicates.

-- Approach 1

SELECT

DISTINCT city

FROM

station

WHERE

RIGHT(city , 1) IN ('a','e','i','o','u');

-- Approach 2

SELECT

DISTINCT city

FROM

station

WHERE

SUBSTR(city,-1,1) IN ('A','E','I','O','U');

-- Approach 3

SELECT

DISTINCT city

FROM

station

WHERE

city RLIKE '.\*[aeiouAEIOU]$';

-- Approach 4

SELECT

DISTINCT city

FROM

station

WHERE

city REGEXP '.\*[aeiouAEIOU]$';

-- Approach 5

SELECT

DISTINCT city

FROM

station

WHERE

city REGEXP '[aeiou]$';

-- Q13. Query the list of CITY names from station that do not start with vowels.

-- Your result cannot contain duplicates.

-- Approach 1

SELECT

DISTINCT city

FROM

station

WHERE

LEFT(city , 1) NOT IN ('a','e','i','o','u');

-- Approach 2

SELECT

DISTINCT city

FROM

station

WHERE

SUBSTR(city,1,1) NOT IN ('A','E','I','O','U');

-- Approach 3

SELECT

DISTINCT city

FROM

station

WHERE

city NOT RLIKE '^[aeiouAEIOU]';

-- Approach 4

SELECT

DISTINCT city

FROM

station

WHERE

city NOT REGEXP '^[aeiouAEIOU]';

-- Approach 5

SELECT

DISTINCT city

FROM

station

WHERE

city NOT REGEXP '^[aeiou]';

-- Q14. Query the list of CITY names from station that do not end with vowels.

-- Your result cannot contain duplicates.

-- Approach 1

SELECT

DISTINCT city

FROM

station

WHERE

RIGHT(city , 1) NOT IN ('a','e','i','o','u');

-- Approach 2

SELECT

DISTINCT city

FROM

station

WHERE

SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');

-- Approach 3

SELECT

DISTINCT city

FROM

station

WHERE

city NOT RLIKE '.\*[aeiouAEIOU]$';

-- Approach 4

SELECT

DISTINCT city

FROM

station

WHERE

city NOT REGEXP '.\*[aeiouAEIOU]$';

-- Approach 5

SELECT

DISTINCT city

FROM

station

WHERE

city NOT REGEXP '[aeiou]$';

-- 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.

-- Approach 1

SELECT

DISTINCT city

FROM

station

WHERE

SUBSTR(city,1,1) NOT IN ('A','E','I','O','U')

OR

SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');

-- Approach 2

SELECT

DISTINCT city

FROM

station

WHERE

city RLIKE '^[^aeiouAEIOU].\*|.\*[^AEIOUaeiou]$';

-- 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.

-- Approach 1

SELECT

DISTINCT city

FROM

station

WHERE

SUBSTR(city,1,1) NOT IN ('A','E','I','O','U')

AND

SUBSTR(city,-1,1) NOT IN ('A','E','I','O','U');

-- Approach 2

SELECT

DISTINCT city

FROM

station

WHERE

city NOT REGEXP '^[aeiou]|[aeiou]$';

-- Approach 3

SELECT

DISTINCT city

FROM

station

WHERE

city REGEXP '^[^aeiou].\*[^aeiou]$';

-- Q.17 Write an SQL query that reports the products that were only sold in the first quarter of 2019.

-- That is, between 2019-01-01 and 2019-03-31 inclusive. Return the result table in any order.

CREATE TABLE product(

product\_id INT,

product\_name VARCHAR(20),

unit\_price INT,

CONSTRAINT prime\_key PRIMARY KEY(product\_id)

);

INSERT INTO product VALUES

(1,'S8',1000),

(2,'G4',800),

(3,'iPhone',1400);

CREATE TABLE sales(

seller\_id INT,

product\_id INT,

buyer\_id INT,

sales\_date DATE,

quantity INT,

price INT,

CONSTRAINT foriegn\_key FOREIGN KEY(product\_id) REFERENCES product(product\_id)

);

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);

-- Approach 1

SELECT

p.product\_id,

p.product\_name

FROM

product p

INNER JOIN

sales s

ON

p.product\_id = s.product\_id

WHERE

p.product\_id

NOT IN (

SELECT

product\_id

FROM

sales

WHERE

sales\_date > '2019-03-31'

)

GROUP BY

p.product\_id

HAVING

MAX(s.sales\_date) <='2019-03-31'

AND

MIN(s.sales\_date) >='2019-01-01';

-- Approach 2

SELECT

p.product\_id,

p.product\_name

FROM

product p

INNER JOIN

sales s

ON

p.product\_id = s.product\_id

GROUP BY

p.product\_id,

p.product\_name

HAVING

MIN(s.sales\_date) >= '2019-01-01' AND MAX(s.sales\_date) <= '2019-03-31';

-- Q.18 Write an SQL query to find all the authors that viewed at least one of their own articles.

-- Return the result table sorted by id in ascending order.

CREATE TABLE views(

article\_id INT,

author\_id INT,

viewer\_id INT,

view\_date DATE

);

INSERT INTO views VALUES

(1,3,5,'2019-08-01'),

(1,3,6,'2019-08-02'),

(2,7,7,'2019-08-01'),

(2,7,6,'2019-08-02'),

(4,7,1,'2019-08-22'),

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

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

SELECT

DISTINCT a.author\_id

FROM

views a

JOIN

views v

ON

a.author\_id = v.viewer\_id

ORDER BY

a.author\_id;

-- Q.19 Write an SQL query to find the percentage of immediate orders in the table,

-- rounded to 2 decimal places.

CREATE TABLE delivery(

delivery\_id INT,

customer\_id INT,

order\_date DATE,

customer\_preferred\_delivery\_date DATE

);

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,'2019-07-11','2019-08-13');

SELECT

ROUND(immediate\_orders \* 100 / total\_orders, 2) AS immediate\_orders\_perct

FROM

(

SELECT

COUNT(

CASE

WHEN order\_date = customer\_preferred\_delivery\_date

THEN customer\_id

END

) AS immediate\_orders,

COUNT(delivery\_id) AS total\_orders

FROM

delivery

) temp ;

-- 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.

CREATE TABLE ads(

ad\_id INT,

user\_id INT,

action ENUM('CLICKED','viewED','IGNORED'),

CONSTRAINT prime\_key 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

ad\_id,

CASE

WHEN (num\_of\_clicks \* 100) / (num\_of\_clicks + num\_of\_views) IS NULL THEN 0

ELSE ROUND((num\_of\_clicks \* 100) / (num\_of\_clicks + num\_of\_views), 2)

END AS ctr

FROM

(

SELECT

ad\_id,

COUNT(

CASE

WHEN action = 'CLICKED' THEN ad\_id

END

) AS num\_of\_clicks,

COUNT(

CASE WHEN action = 'VIEWED' THEN ad\_id

END

) AS num\_of\_views

FROM

ads

GROUP BY

ad\_id

) temp\_ads

ORDER BY

ctr DESC,

ad\_id ASC;

-- Q.21 Write an SQL query to find the team size of each of the employees.

CREATE TABLE employee(

employee\_id INT,

team\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(employee\_id)

);

INSERT INTO employee VALUES

(1,8),

(2,8),

(3,8),

(4,7),

(5,9),

(6,9);

SELECT

employee\_id,

COUNT(employee\_id) OVER(PARTITION BY team\_id) AS team\_size

FROM

employee

ORDER BY

employee\_id;

-- 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.

CREATE TABLE countries(

country\_id INT,

country\_name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(country\_id)

);

INSERT INTO countries VALUES

(2, 'USA'),

(3, 'AUSTRALIA'),

(7, 'PERU'),

(5, 'CHINA'),

(8, 'MOROCCO'),

(9, 'SPAIN');

CREATE TABLE weather(

country\_id INT,

weather\_state INT,

day DATE,

CONSTRAINT prime\_key PRIMARY KEY(country\_id, day)

);

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,'2019-10-09'),

(5,21,'2019-10-23'),

(7,25,'2019-11-08'),

(7,22,'2019-12-01'),

(7,20,'2019-12-02'),

(8,25,'2019-11-05'),

(8,27,'2019-11-15'),

(8,31,'2019-11-25'),

(9,7,'2019-10-23'),

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

SELECT

c.country\_name,

CASE

WHEN AVG(weather\_state) <= 15 THEN 'COLD'

WHEN AVG(weather\_state) >= 25 THEN 'HOT'

ELSE 'WARM'

END AS avg\_weather

FROM

countries c

INNER JOIN

weather w

ON

c.country\_id = w.country\_id

WHERE

day BETWEEN '2019-11-01' AND '2019-11-30'

GROUP BY

c.country\_id,

c.country\_name;

-- Q.23 Write an SQL query to find the average selling price for each product.

-- average\_price should be rounded to 2 decimal places.

CREATE TABLE prices(

product\_id INT,

start\_date DATE,

end\_date DATE,

price INT,

CONSTRAINT prime\_key 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);

CREATE TABLE units\_sold(

product\_id INT,

purchase\_date DATE,

units INT

);

INSERT INTO units\_sold VALUES

(1,'2019-02-25',100),

(1,'2019-03-01',15),

(2,'2019-02-10',200),

(2,'2019-02-22',30);

SELECT

p.product\_id,

ROUND(SUM(p.price \* u.units)/SUM(u.units), 2) AS average\_price

FROM

units\_sold u

INNER JOIN

prices p

ON

p.product\_id = u.product\_id

WHERE

u.purchase\_date BETWEEN p.start\_date AND p.end\_date

GROUP BY

p.product\_id;

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

-- Return the result table in any order.

CREATE TABLE activity(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT,

CONSTRAINT prime\_key PRIMARY KEY(player\_id, event\_date)

);

INSERT INTO activity VALUES

(1,2,'2016-03-01',5),

(1,2,'2016-03-02',6),

(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),

(3,4,'2018-07-03',5);

-- Approach 1

WITH temp\_activity AS (

SELECT

player\_id,

event\_date as first\_login,

ROW\_NUMBER() OVER(PARTITION BY player\_id ORDER BY event\_date) as ranking

FROM

activity

)

SELECT

player\_id,

first\_login

FROM

temp\_activity

WHERE

ranking = 1;

-- Approach 2

SELECT

player\_id,

first\_login

FROM (

SELECT

player\_id,

event\_date as first\_login,

ROW\_NUMBER() OVER(PARTITION BY player\_id ORDER BY event\_date) ranking

FROM

activity

) temp\_activity

WHERE

ranking = 1;

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

-- Return the result table in any order.

-- Approach 1

WITH temp\_activity AS (

SELECT

player\_id,

device\_id,

ROW\_NUMBER() OVER(PARTITION BY player\_id ORDER BY device\_id) ranking

FROM

activity

)

SELECT

player\_id,

device\_id

FROM

temp\_activity

WHERE

ranking = 1;

-- Approach 2

SELECT

player\_id,

device\_id

FROM (

SELECT

player\_id,

device\_id,

ROW\_NUMBER() OVER(PARTITION BY player\_id ORDER BY device\_id) as ranking

FROM

activity

) temp\_activity

WHERE

ranking = 1;

-- 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.

CREATE TABLE products(

product\_id INT,

product\_name VARCHAR(30),

product\_category VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(product\_id)

);

INSERT INTO products VALUES

(1,'LEETCODE SOLUTIONS','BOOK'),

(2,'JEWELS OF STRINGOLOGY','BOOK'),

(3,'HP','LAPTOP'),

(4,'LENOVO','LAPTOP'),

(5,'LEETCODE KIT','T-SHIRT');

CREATE TABLE orders(

product\_id INT,

order\_date DATE,

unit INT,

CONSTRAINT foriegn\_key FOREIGN KEY(product\_id) REFERENCES products(product\_id)

);

INSERT INTO orders VALUES

(1,'2020-02-05',60),

(1,'2020-02-05',70),

(2,'2020-01-05',30),

(2,'2020-02-05',80),

(3,'2020-02-05',2),

(3,'2020-02-05',3),

(4,'2020-03-05',20),

(4,'2020-03-05',30),

(4,'2020-03-05',60),

(5,'2020-02-05',50),

(5,'2020-02-05',50),

(5,'2020-03-05',50);

SELECT

p.product\_name,

SUM(o.unit) AS total\_unit\_sold

FROM

products p

INNER JOIN

orders o

ON

p.product\_id = o.product\_id

WHERE

order\_date BETWEEN '2020-02-01' AND '2020-02-28'

GROUP BY

p.product\_id

HAVING

SUM(o.unit) >= 100;

-- Q.27 Write an SQL query to find the users who have valid emails.

CREATE TABLE users(

user\_id INT,

name VARCHAR(25),

mail VARCHAR(25),

CONSTRAINT prime\_key PRIMARY KEY(user\_id)

);

INSERT INTO users VALUE

(1,'WINSTON','winston@leetcode.com'),

(2,'JONATHAN','jonathonisgreat'),

(3,'ANNABELLE','bella-@leetcode.com'),

(4,'SALLY','sally.come@leetcode.com'),

(5,'MARWAN','quarz-- 2020@leetcode.com'),

(6,'DAVID','david45@gmail.com'),

(7,'SHAPIRO','.shapo@leetcode.com');

SELECT

user\_id,

name,

mail

FROM

users

WHERE

mail LIKE '%@leetcode.com' AND mail REGEXP '^[a-zA-Z0-9][a-zA-Z0-9.\_-]\*@[a-zA-Z0-9][a-zA-Z0-9.\_-]\*\\.[a-zA-Z]{2,4}$';

-- 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 2020. Return the result table in any order.

CREATE TABLE customers(

customer\_id INT,

name VARCHAR(20),

country VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(customer\_id)

);

CREATE TABLE orders(

order\_id INT,

customer\_id INT,

product\_id INT,

order\_date DATE,

quantity INT,

CONSTRAINT prime\_key PRIMARY KEY(order\_id)

);

CREATE TABLE products(

product\_id INT,

description VARCHAR(20),

price INT,

CONSTRAINT prime\_key PRIMARY KEY(product\_id)

);

INSERT INTO customers VALUES

(1,'WINSTON','USA'),

(2,'JONATHON','PERU'),

(3,'MOUSTAFA','EGYPT');

INSERT INTO products 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

customer\_id,

name

FROM

(

SELECT

c.customer\_id,

c.name,

EXTRACT(MONTH FROM o.order\_date) AS month\_extracted,

SUM(o.quantity \* p.price) AS total\_spent

FROM

orders o

INNER JOIN

customers c

ON

c.customer\_id = o.customer\_id

INNER JOIN

products p

ON

p.product\_id = o.product\_id

WHERE

o.order\_date BETWEEN '2020-06-01' AND '2020-07-31'

GROUP BY

c.customer\_id,

c.name,

EXTRACT(MONTH FROM o.order\_date)

) temp\_customers

WHERE

total\_spent >= 100

GROUP BY

customer\_id

HAVING

COUNT(customer\_id) = 2;

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

-- Return the result table in any order.

CREATE TABLE tv\_program(

program\_date DATETIME,

content\_id INT,

channel VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(program\_date, content\_id)

);

CREATE TABLE content(

content\_id INT,

title VARCHAR(20),

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

content\_type VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(content\_id)

);

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');

INSERT INTO tv\_program VALUES

('2020-06-10 18:00',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

DISTINCT c.title

FROM

content c

INNER JOIN

tv\_program t

ON

c.content\_id = T.content\_id

WHERE

c.kids\_content = 'Y'

AND

c.content\_type = 'movies'

AND

T.program\_date BETWEEN '2020-06-01' AND '2020-06-30';

-- Q.30 Write an SQL query to find the npv of each query of the Queries table.

-- Return the result table in any order.

CREATE TABLE npv(

id INT,

year INT,

npv INT,

CONSTRAINT prime\_key PRIMARY KEY(id, year)

);

CREATE TABLE queries(

id INT,

year INT,

CONSTRAINT prime\_key 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

DISTINCT n.id,

n.year,

n.npv

FROM

queries q

INNER JOIN

npv n

ON

n.id = q.id

AND

n.year = q.year;

-- Q.31 SAME AS 30

-- 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. Return the result table in any order.

CREATE TABLE employees(

id INT,

name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(id)

);

CREATE TABLE employees\_uni(

id INT,

unique\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(id, unique\_id)

);

INSERT INTO employees VALUES

(1,'ALICE'),

(7,'BOB'),

(11,'MEIR'),

(90,'WINSTON'),

(3,'JONATHAN');

INSERT INTO employees\_uni VALUES

(3,1),

(11,2),

(90,3);

SELECT

eu.unique\_id,

e.name

FROM

employees e

LEFT JOIN

employees\_uni eu

ON

e.id = eu.id

ORDER BY

e.name;

-- 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.

CREATE TABLE users(

id INT,

name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(id)

);

INSERT INTO users VALUES

(1,'ALICE'),

(2,'BOB'),

(3,'ALEX'),

(4,'DONALD'),

(7,'LEE'),

(13,'JONATHON'),

(19,'ELVIS');

CREATE TABLE rides(

id INT,

user\_id INT,

distance INT,

CONSTRAINT prime\_key PRIMARY KEY(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

u.name AS riders,

COALESCE(SUM(r.distance), 0) AS distance\_travelled

FROM

users u

LEFT JOIN

rides r

ON

u.id = r.user\_id

GROUP BY

u.name

ORDER BY

distance\_travelled DESC,

riders;

-- Q.34 SAME AS 26

-- 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.

CREATE TABLE users(

user\_id INT,

name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(user\_id)

);

INSERT INTO users VALUES

(1,'DANIEL'),

(2,'MONICA'),

(3,'MARIA'),

(4,'JAMES');

CREATE TABLE movies(

movie\_id INT,

title VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(movie\_id)

);

INSERT INTO movies VALUES

(1,'AVENGERS'),

(2,'FROZEN 2'),

(3,'JOKER');

CREATE TABLE movie\_rating(

movie\_id INT,

user\_id INT,

rating INT,

created\_at DATE,

CONSTRAINT prime\_key PRIMARY KEY(movie\_id, user\_id)

);

INSERT INTO movie\_rating 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');

-- Find the name of the user who has rated the greatest number of movies

SELECT

u.name as user

FROM

users u

INNER JOIN

movie\_rating mr

ON

u.user\_id = mr.user\_id

GROUP BY

u.user\_id

ORDER BY

COUNT(u.name) DESC,

LENGTH(u.name)

LIMIT 1;

-- Find the movie name with the highest average rating in February 2020.

SELECT

m.title

FROM

movies m

INNER JOIN

movie\_rating mr

ON

m.movie\_id = mr.movie\_id

WHERE

mr.created\_at BETWEEN '2020-02-01' AND '2020-02-28'

GROUP BY

m.movie\_id

ORDER BY

AVG(rating) DESC,

m.title

LIMIT 1;

-- Q.36 SAME AS 33

-- Q.37 SAME AS 32

-- 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. Return the result table in any order.

CREATE TABLE departments(

id INT,

name VARCHAR(25),

CONSTRAINT prime\_key PRIMARY KEY(id)

);

INSERT INTO departments VALUES

(1,'ELECTRICAL ENGINEERING'),

(7,'COMPUTER ENGINEERING'),

(13,'BUSINESS ADMINISTRATION');

CREATE TABLE students(

id INT,

name VARCHAR(25),

department\_id INT,

CONSTRAINT prime\_key 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,'JONATHON',7),

(7,'DAIANA',33),

(11,'MADELYNN',1);

SELECT

id,

name

FROM

students

WHERE

department\_id NOT IN (

SELECT

id

FROM

departments

);

-- 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.

-- Return the result table in any order.

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);

-- Approach 1

SELECT

CASE

WHEN from\_id < to\_id THEN from\_id

ELSE to\_id

END AS person1,

CASE

WHEN from\_id < to\_id THEN to\_id

ELSE from\_id

END AS person2,

COUNT(\*) AS call\_count,

SUM(duration) AS total\_duration

FROM

calls

GROUP BY

person1,

person2;

-- Approach 2

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

person1,

person2;

-- Q.40 SAME AS 23

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

-- occupies in each warehouse. Return the result table in any order.

CREATE TABLE warehouse(

name VARCHAR(25),

product\_id INT,

units INT,

CONSTRAINT prime\_key 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);

CREATE TABLE products(

product\_id INT,

product\_name VARCHAR(25),

width INT,

length INT,

height INT,

CONSTRAINT prime\_key PRIMARY KEY(product\_id)

);

INSERT INTO products VALUES

(1,'LC-TV',5,50,40),

(2,'LC-KEYCHAIN',5,5,5),

(3,'LC-PHONE',2,10,10),

(4,'LC-SHIRT',4,10,20);

SELECT

w.name AS warehouse\_name,

SUM(p.length \* p.width \* p.height \* w.units) AS volume

FROM

warehouse w

INNER JOIN

products p

ON

w.product\_id = p.product\_id

GROUP BY

w.name;

-- 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.

CREATE TABLE sales(

sale\_date DATE,

fruit ENUM('APPLES','ORANGES'),

sold\_num INT,

CONSTRAINT prime\_key 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);

-- Approach 1

SELECT

sale\_date,

difference

FROM

(

SELECT

sale\_date,

sold\_num - LEAD(sold\_num, 1) OVER(PARTITION BY sale\_date) AS difference

FROM

sales

) temp\_sales

WHERE

difference IS NOT NULL

ORDER BY

sale\_date;

-- Approach 2

SELECT

sale\_date,

SUM(

CASE

WHEN fruit = 'APPLES' THEN sold\_num

WHEN fruit = 'ORANGES' THEN -sold\_num

END

) AS difference

FROM

sales

GROUP BY

sale\_date

ORDER BY

sale\_date;

-- Approach 3

SELECT

s.sale\_date,

s.sold\_num - ss.sold\_num AS difference

FROM

sales s

INNER JOIN

sales ss

ON

s.sale\_date = ss.sale\_date

WHERE

s.fruit = 'APPLES' AND ss.fruit = 'ORANGES'

ORDER BY

sale\_date;

-- Approach 4

SELECT

sale\_date,

SUM(IF(fruit = 'APPLES', sold\_num, -sold\_num)) AS difference

FROM

sales

GROUP BY

sale\_date

ORDER BY

sale\_date;

-- Approach 5

SELECT

sale\_date,

SUM(

CASE

WHEN fruit = 'APPLES' THEN sold\_num

ELSE -sold\_num

END

) AS difference

FROM

sales

GROUP BY

sale\_date

ORDER BY

sale\_date;

-- 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.

-- Approach 1

WITH temp\_activity AS (

SELECT

player\_id,

LEAD(event\_date, 1) OVER(PARTITION BY player\_id ORDER BY event\_date) - event\_date AS difference

FROM

activity

),

temp\_activity2 AS (

SELECT

COUNT(DISTINCT player\_id) AS players\_count

FROM

temp\_activity

WHERE

difference = 1

GROUP BY

player\_id

)

SELECT

ROUND(COUNT(\*) / (SELECT COUNT(DISTINCT player\_id) FROM activity), 2) AS fraction

FROM

temp\_activity2;

-- Approach 2

WITH temp\_activity AS (

SELECT

player\_id,

event\_date,

DATEDIFF(event\_date,lag(event\_date) OVER(PARTITION BY player\_id ORDER BY event\_date)) AS difference

FROM

activity

),

temp\_activity2 AS (

SELECT

COUNT(DISTINCT player\_id) AS players\_count

FROM

temp\_activity

WHERE

difference = 1

GROUP BY

player\_id

)

SELECT

ROUND(COUNT(\*) / (SELECT COUNT(DISTINCT player\_id) FROM activity), 2) AS fraction

FROM

temp\_activity2;

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

CREATE TABLE employee(

id INT,

name VARCHAR(20),

department VARCHAR(20),

manager\_id INT,

CONSTRAINT prime\_key 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','A',101),

(107,'BUTTLER','A',111),

(108,'JIMMY','A',121),

(111,'ROOT','A',NULL),

(121,'POPE','A',NULL);

-- Approach 1

SELECT

name

FROM

employee

WHERE

id = (

SELECT

e.manager\_id

FROM

employee e

INNER JOIN

employee ee

ON

e.manager\_id = ee.id

GROUP BY

e.manager\_id

HAVING

COUNT(e.manager\_id) >= 5

);

-- Approach 2

SELECT

name

FROM

employee

WHERE

id = (

SELECT

manager\_id

FROM

employee

GROUP BY

manager\_id

HAVING

COUNT(manager\_id) >= 5

);

-- 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

CREATE TABLE department(

dept\_id INT,

department\_name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(dept\_id)

);

CREATE TABLE student(

student\_id INT,

student\_name VARCHAR(20),

gender VARCHAR(6),

dept\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(student\_id),

CONSTRAINT foriegn\_key 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.department\_name,

COUNT(s.student\_id) AS number\_of\_students

FROM

department d

LEFT JOIN

student s

ON

d.dept\_id = s.dept\_id

GROUP BY

d.department\_name

ORDER BY

number\_of\_students DESC,

department\_name;

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

-- all the products in the product table. Return the result table in any order.

CREATE TABLE customer(

customer\_id INT,

product\_key INT

);

INSERT INTO customer VALUES

(1,5),

(2,6),

(3,5),

(3,6),

(1,6);

CREATE TABLE product(

product\_key INT,

CONSTRAINT prime\_key PRIMARY KEY(product\_key)

);

INSERT INTO product VALUES

(5),

(6);

SELECT

customer\_id

FROM

customer

GROUP BY

customer\_id

HAVING

COUNT(DISTINCT product\_key) = (

SELECT

COUNT(DISTINCT product\_key)

FROM

product

);

-- Q.47 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.

CREATE TABLE employee(

employee\_id INT,

name VARCHAR(20),

experience\_years INT,

CONSTRAINT prime\_key PRIMARY KEY(employee\_id)

);

CREATE TABLE project(

project\_id INT,

employee\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(project\_id, 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);

-- Approach 1

SELECT

project\_id,

employee\_id

FROM

(

SELECT

p.project\_id,

e.employee\_id,

DENSE\_RANK() OVER(PARTITION BY p.project\_id ORDER BY e.experience\_years DESC) AS ranking

FROM

employee e

INNER JOIN

project p

ON

e.employee\_id = p.employee\_id

) temp\_employee

WHERE

ranking = 1;

-- Approach 2

SELECT

p.project\_id,

e.employee\_id

FROM

employee e

INNER JOIN

project p

ON

e.employee\_id = p.employee\_id

WHERE

e.experience\_years =

(

SELECT

MAX(experience\_years)

FROM

employee

);

-- 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.

-- Return the result table in any order.

CREATE TABLE books(

book\_id INT,

name VARCHAR(20),

available\_from DATE,

CONSTRAINT prime\_key PRIMARY KEY(book\_id)

);

CREATE TABLE orders(

order\_id INT,

book\_id INT,

quantity INT,

dispatch\_date DATE,

CONSTRAINT prime\_key PRIMARY KEY(order\_id),

CONSTRAINT foriegn\_key FOREIGN KEY(book\_id) REFERENCES books(book\_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');

INSERT INTO orders 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

b.book\_id, b.name

FROM

books b

LEFT JOIN

orders o

ON

b.book\_id = o.book\_id

WHERE

available\_from < '2019-05-23'

AND

(o.dispatch\_date BETWEEN '2018-06-23' AND '2019-06-23')

OR

dispatch\_date IS NULL

GROUP BY

b.book\_id,

b.name

HAVING

COALESCE(SUM(o.quantity), 0) < 10;

-- 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.

CREATE TABLE enrollments(

student\_id INT,

course\_id INT,

grade INT,

CONSTRAINT prime\_key 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);

-- Approach 1

SELECT

student\_id,

course\_id,

grade

FROM

(

SELECT

student\_id,

course\_id,

grade,

ROW\_NUMBER() OVER(PARTITION BY student\_id ORDER BY grade DESC) AS ranking

FROM

enrollments

) temp\_enrollments

WHERE

ranking = 1

order by

student\_id;

-- Approach 2

WITH temp\_enrollments AS

(

SELECT

student\_id,

course\_id,

grade,

ROW\_NUMBER() OVER(PARTITION BY student\_id ORDER BY grade DESC) AS ranking

FROM

enrollments

)

SELECT

student\_id,

course\_id,

grade

FROM

temp\_enrollments

WHERE

ranking = 1

order by

student\_id;

-- Q.50 Write an SQL query to find the winner in each group. Return the result table in any order.

CREATE TABLE matches(

match\_id INT,

first\_player INT,

second\_player INT,

first\_player\_goals INT,

second\_player\_goals INT,

CONSTRAINT prime\_key PRIMARY KEY(match\_id)

);

CREATE TABLE players(

player\_id INT,

group\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(player\_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);

INSERT INTO players VALUES

(15,1),

(25,1),

(30,1),

(45,1),

(10,2),

(35,2),

(50,2),

(20,3),

(40,3);

SELECT

group\_id,

players as player\_id

FROM (

SELECT

p.group\_id,

CASE

WHEN first\_player\_goals > second\_player\_goals THEN first\_player

WHEN first\_player\_goals < second\_player\_goals THEN second\_player

WHEN first\_player\_goals = second\_player\_goals THEN IF(first\_player < second\_player, first\_player, second\_player)

END AS players,

MAX(IF(first\_player\_goals > second\_player\_goals, first\_player\_goals, second\_player\_goals)) AS goals,

ROW\_NUMBER() OVER(PARTITION BY team\_id ORDER BY MAX(IF(first\_player\_goals > second\_player\_goals, first\_player\_goals, second\_player\_goals)) DESC) AS ranking

FROM

players p

INNER JOIN

matches m

ON

m.first\_player = p.player\_id

OR

m.second\_player = p.player\_id

GROUP BY

p.group\_id,

players

) temp\_matches

WHERE

ranking = 1;

-- Q.51 Write an SQL Query to report the name, population, and area of the big countries.

-- Return the result table in any order .

CREATE TABLE world(

name VARCHAR(20) NOT NULL,

continent VARCHAR(15) NOT NULL,

area INT NOT NULL,

population BIGINT NOT NULL,

gdp BIGINT NOT NULL,

CONSTRAINT prime\_key PRIMARY KEY(name)

);

INSERT INTO world VALUES

('Afghanistan', 'Asia', 652230, 25500100, 203430000000),

('Albania', 'Europe', 28748, 2831741, 12960000000),

('Algeria', 'Africa', 2381741, 37100000, 188681000000),

('Andorra', 'Europe', 468, 78115, 3712000000),

('Angola', 'Africa', 1246700, 20609294, 100990000000),

('Dominican Republic', 'Caribbean', 48671, 9445281, 58898000000),

('China', 'Asia', 652230, 1365370000, 8358400000000),

('Colombia', 'South America', 1141748, 47662000, 369813000000),

('Comoros', 'Africa', 1862, 743798, 616000000),

('Denmark', 'Europe', 43094, 5634437, 314889000000),

('Djibouti', 'Africa', 23200, 886000, 1361000000),

('Dominica', 'Caribbean', 751, 71293, 499000000),

('SriLanka', 'Asia', 652230, 25500100, 203430000000);

SELECT

name,

population,

area

FROM

world

WHERE

area > 3000000

OR

population > 25000000;

-- Q.52 Write an SQL Query to report the names of the customer that are not referred by the customer with id = 2.

-- Return the result table in any order.

CREATE TABLE customer(

id INT,

name VARCHAR(10),

refree\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(id)

);

INSERT INTO customer VALUES

(1,'Will',NULL),

(2,'Jane',NULL),

(3,'Alex',2),

(4,'Bill',NULL),

(5,'Zack',1),

(6,'Mark',2);

SELECT

name

FROM

customer

WHERE

refree\_id <> 2

OR

refree\_id IS NULL;

-- Q.53 Write an SQL Query to report all customers who never order anything.

-- Return the result table in any order .

CREATE TABLE orders(

id INT,

customer\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(id)

);

INSERT INTO orders VALUES

(1,3),

(2,1);

CREATE TABLE customers(

id INT,

name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(id)

);

INSERT INTO customers VALUES

(1,'JOE'),

(2,'HENRY'),

(3,'SAM'),

(4,'MAX');

SELECT

name AS customers

FROM

customers

WHERE

id NOT IN (

SELECT

customer\_id

FROM

orders

);

-- Q.54 Write an SQL Query to find the team size of each of the employees.

-- Return result table in any order .

CREATE TABLE employee(

employee\_id INT,

team\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(employee\_id)

);

INSERT INTO employee VALUES

(1,8),

(2,8),

(3,8),

(4,7),

(5,9),

(6,9);

SELECT

employee\_id,

COUNT(employee\_id) OVER(PARTITION BY team\_id ORDER BY employee\_id

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS team\_size

FROM

employee

ORDER BY

employee\_id;

-- Q.55 Write an SQL Query to find the countries where this company can invest .

-- Return the result table in any order .

CREATE TABLE person(

id INT,

name VARCHAR(20),

phone\_number VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(id)

);

CREATE TABLE country(

name VARCHAR(20),

country\_code VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(country\_code)

);

CREATE TABLE calls(

caller\_id INT,

callee\_id INT,

duration INT

);

INSERT INTO person VALUES

(3,'JONATHON','051-1234567'),

(21,'ELVIS','051-7654321'),

(1,'MONCEF','212-1234567'),

(2,'MAROUA','212-6523651'),

(7,'MEIR','972-1234567'),

(9,'RACHEL','972-0011100');

INSERT INTO calls VALUES

(1,9,33),

(1,2,59),

(3,12,102),

(3,12,330),

(12,3,5),

(7,9,13),

(7,1,3),

(9,7,1),

(1,7,7),

(2,9,4);

INSERT INTO country VALUES

('PERU','51'),

('ISRAEL','972'),

('MOROCCO','212'),

('GERMANY','49'),

('ETHIOPIA','251');

-- Approach 1

SELECT

name AS country

FROM (

SELECT

c.name,

SUM(ca.duration) AS call\_duration,

COUNT(c.country\_code) AS number\_of\_calls

FROM (

SELECT

id,

name,

CASE

WHEN LEFT(SUBSTR(phone\_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone\_number, 1,3), (LENGTH(SUBSTR(phone\_number, 1,3))-1))

ELSE SUBSTR(phone\_number, 1,3) END AS country\_code

FROM

person

) temp\_person

JOIN

country c

ON

temp\_person.country\_code = c.country\_code

JOIN

calls ca

ON

temp\_person.id = caller\_id

GROUP BY

c.name

UNION ALL

SELECT

c.name,

SUM(ca.duration) AS call\_duration,

COUNT(c.country\_code) AS number\_of\_calls

FROM (

SELECT

id,

name,

CASE

WHEN LEFT(SUBSTR(phone\_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone\_number, 1,3), (length(SUBSTR(phone\_number, 1,3))-1))

ELSE SUBSTR(phone\_number, 1,3) END AS country\_code

FROM

person

) temp\_person

JOIN

country c

ON

temp\_person.country\_code = c.country\_code

JOIN

calls ca

ON

temp\_person.id = ca.callee\_id

GROUP BY

c.name

) temp

GROUP BY

name

HAVING

SUM(call\_duration)/SUM(number\_of\_calls) > (SELECT AVG(duration) FROM calls);

-- Approach 2

WITH temp\_person AS (

SELECT

id,

name,

CASE

WHEN LEFT(SUBSTR(phone\_number, 1,3),1) = '0' THEN RIGHT(SUBSTR(phone\_number, 1,3), (LENGTH(SUBSTR(phone\_number, 1,3))-1))

ELSE SUBSTR(phone\_number, 1,3) END AS country\_code

FROM

person

)

SELECT

name AS country

FROM (

SELECT

c.name,

SUM(ca.duration) AS call\_duration,

COUNT(c.country\_code) AS number\_of\_calls

FROM

temp\_person

JOIN

country c

ON

temp\_person.country\_code = c.country\_code

JOIN

calls ca

ON

temp\_person.id = ca.caller\_id

GROUP BY

c.name

UNION ALL

SELECT

c.name,

SUM(ca.duration) AS call\_duration,

COUNT(c.country\_code) AS number\_of\_calls

FROM

temp\_person

JOIN

country c

ON

temp\_person.country\_code = c.country\_code

JOIN

calls ca

ON

temp\_person.id = ca.callee\_id

GROUP BY

c.name

)temp

GROUP BY

name

HAVING

SUM(call\_duration)/SUM(number\_of\_calls) > (SELECT AVG(duration) FROM calls);

-- Q.56 Write an SQL Query to report the device that is first logged in for each player.

-- Return the result table in any order.

CREATE TABLE activity(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT,

CONSTRAINT prime\_key PRIMARY KEY(player\_id, event\_date)

);

INSERT INTO activity VALUES

(1,2,'2016-03-01',5),

(1,2,'2016-03-02',6),

(2,3,'2017-06-25',1),

(3,1,'2016-03-02',0),

(3,4,'2018-07-03',5);

SELECT

player\_id,

device\_id

FROM

(

SELECT

player\_id,

device\_id,

event\_date,

ROW\_number() OVER(PARTITION BY player\_id ORDER BY event\_date) ranking

FROM

activity

) temp\_activity

WHERE

ranking = 1;

-- Q.57 Write an SQL Query to find the customer\_number for the customer who has placed the largest number of orders.

CREATE TABLE orders(

order\_number INT,

customer\_number INT,

CONSTRAINT prime\_key PRIMARY KEY(order\_number)

);

INSERT INTO orders VALUES

(1,1),

(2,2),

(3,3),

(4,3);

WITH temp\_orders AS (

SELECT

DISTINCT customer\_number,

DENSE\_RANK() OVER(ORDER BY total\_orders DESC) AS ranking

FROM (

SELECT

customer\_number,

COUNT(order\_number) OVER(PARTITION BY customer\_number) total\_orders

FROM

orders

) temp\_cust\_details

)

SELECT

customer\_number

FROM

temp\_orders

WHERE

ranking = 1;

-- Q.58 Write an SQL Query to report all the consecutive available seats in the cinema.

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

CREATE TABLE cinema(

seat\_id INT AUTO\_INCREMENT,

free BOOLEAN,

CONSTRAINT prime\_key PRIMARY KEY(seat\_id)

);

INSERT INTO cinema (free) VALUES

(1),(0),(1),(1),(1),(1),(0),(1),

(1),(0),(1),(1),(1),(0),(1),(1);

SELECT

DISTINCT c1.seat\_id

FROM

cinema c1

INNER JOIN

cinema c2

ON

ABS(c1.seat\_id - c2.seat\_id) = 1

AND

(c1.free = 1 AND c2.free = 1)

ORDER BY

c1.seat\_id;

-- Q.59 Write an SQL Query to report the names of all the salespersons who did not have any

-- orders related to the company with the name "RED".

CREATE TABLE sales\_person(

sales\_id INT,

name VARCHAR(20),

salary INT,

commission\_rate INT,

hire\_date VARCHAR(25),

CONSTRAINT prime\_key PRIMARY KEY(sales\_id)

);

INSERT INTO sales\_person VALUES

(1,'JOHN',100000,6,'4/1/2006'),

(2,'AMY',12000,5,'5/1/2010'),

(3,'MARK',65000,12,'12/25/2008'),

(4,'PAM',25000,25,'1/1/2005'),

(5,'ALEX',5000,10,'2/3/2007');

CREATE TABLE company(

company\_id INT,

name VARCHAR(20),

city VARCHAR(10),

CONSTRAINT prime\_key PRIMARY KEY(company\_id)

);

INSERT INTO company VALUES

(1,'RED','BOSTON'),

(2,'ORANGE','NEW YORK'),

(3,'YELLOW','BOSTON'),

(4,'GREEN','AUSTIN');

CREATE TABLE orders(

order\_id INT,

order\_date VARCHAR(30),

company\_id INT,

sales\_id INT,

amount INT,

CONSTRAINT prime\_key PRIMARY KEY(order\_id),

CONSTRAINT company\_foreign\_key FOREIGN KEY (company\_id) REFERENCES company(company\_id),

CONSTRAINT sales\_foreign\_key FOREIGN KEY (sales\_id) REFERENCES sales\_person(sales\_id)

);

INSERT INTO orders VALUES

(1,'1/1/2014',3,4,10000),

(2,'2/1/2014',4,5,5000),

(3,'3/1/2014',1,1,50000),

(4,'4/1/2014',1,4,25000);

SELECT

name

FROM

sales\_person

WHERE

sales\_id NOT IN (

SELECT

o.sales\_id

FROM

orders o

INNER JOIN

company c

ON

c.company\_id = o.company\_id

WHERE

c.name = 'RED'

);

-- Q.60 Write an SQL Query to report for every three line segments whether they can form a triangle.

-- Return the result table in any order.

CREATE TABLE triangle(

x INT,

y INT,

z INT,

CONSTRAINT prime\_key PRIMARY KEY(x,y,z)

);

INSERT INTO triangle VALUES

(13,15,30),

(10,20,15);

SELECT

x,

y,

z,

IF(x+y>z AND x+z>y AND y+z>x, 'YES','NO') AS is\_triangle

FROM

triangle;

-- Q.61 Write an SQL Query to report the shortest distance between any two points from the Point table.

CREATE TABLE point(

x INT,

CONSTRAINT prime\_key PRIMARY KEY(x)

);

INSERT INTO point VALUES

(-1),

(0),

(2);

SELECT

MIN(ABS(c1.x - c2.x)) AS shortest\_distance

FROM

point c1

INNER JOIN

point c2

WHERE

c1.x!=c2.x;

-- Q.62 Write a SQL Query for a report that provides the pairs (actor\_id, director\_id) where the actor has

-- cooperated with the director at least three times. Return the result table in any order.

CREATE TABLE actor\_director(

actor\_id INT,

director\_id INT,

timestamp INT,

CONSTRAINT prime\_key PRIMARY KEY(timestamp)

);

INSERT INTO actor\_director VALUES

(1,1,0),

(1,1,1),

(1,1,2),

(1,2,3),

(1,2,4),

(2,1,5),

(2,1,6);

WITH temp\_actor\_director AS (

SELECT

DISTINCT actor\_id,

director\_id,

DENSE\_RANK() OVER(ORDER BY total\_movies DESC) AS ranking

FROM (

SELECT

actor\_id,

director\_id,

COUNT(actor\_id) OVER(PARTITION BY actor\_id, director\_id) AS total\_movies

FROM

actor\_director

) temp

)

SELECT

actor\_id,

director\_id

FROM

temp\_actor\_director

WHERE

ranking = 1;

-- Q.63 Write an SQL Query that reports the product\_name, year, and price for each sale\_id in

-- the sales table. Return the resulting table in any order.

CREATE TABLE sales(

sale\_id INT,

product\_id INT,

year INT,

Quantity INT,

price INT,

CONSTRAINT prime\_key PRIMARY KEY(sale\_id, year)

);

CREATE TABLE product(

product\_id INT,

product\_name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(product\_id)

);

INSERT INTO sales VALUES

(1,100,2008,10,5000),

(2,100,2009,12,5000),

(7,200,2011,15,9000);

INSERT INTO product VALUES

(100,'NOKIA'),

(200,'APPLE'),

(300,'SAMSUNG');

SELECT

p.product\_name,

s.year,

s.price

FROM

sales s

INNER JOIN

product p

ON

p.product\_id = s.product\_id;

-- Q.64 Write an SQL Query that reports the average experience years of all the employees for each project,

-- rounded to 2 digits. Return the result table in any order.

CREATE TABLE project(

project\_id INT,

employee\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(project\_id, employee\_id)

);

INSERT INTO project VALUES

(1,1),

(1,2),

(1,3),

(2,1),

(2,4);

CREATE TABLE employee(

employee\_id INT,

name VARCHAR(20),

experience\_years INT,

CONSTRAINT prime\_key PRIMARY KEY(employee\_id)

);

INSERT INTO employee VALUES

(1,'KHALED',3),

(2,'ALI',2),

(3,'JOHN',1),

(4,'DOE',2);

-- Approach 1

SELECT

p.project\_id,

ROUND(AVG(experience\_years), 2) AS average\_years

FROM

employee e

INNER JOIN

project p

ON

p.employee\_id = e.employee\_id

GROUP BY

project\_id;

-- Approach 2

SELECT

DISTINCT p.project\_id,

ROUND(AVG(experience\_years) OVER(PARTITION BY project\_id), 2) AS average\_years

FROM

employee e

INNER JOIN

project p

ON

p.employee\_id = e.employee\_id;

-- Q.65 Write an SQL Query that reports the best seller by total sales price, If there is a tie,

-- report them all. Return the result table in any order.

CREATE TABLE product(

product\_id INT,

product\_name VARCHAR(20),

unit\_price INT,

CONSTRAINT prime\_key PRIMARY KEY(product\_id)

);

INSERT INTO product VALUES

(1,'S8',1000),

(2,'G4',800),

(3,'Iphone',1400);

CREATE TABLE sales(

seller\_id INT,

product\_id INT,

buyer\_id INT,

sale\_date DATE,

quantity INT,

price INT,

CONSTRAINT FOREIGN\_KEY FOREIGN KEY(product\_id) REFERENCES product(product\_id)

);

INSERT INTO sales VALUES

(1,1,1,'2019-01-21',2,2000),

(1,2,2,'2019-01-21',1,800),

(2,2,3,'2019-01-21',1,800),

(3,3,4,'2019-01-21',2,2800);

WITH temp\_sales AS (

SELECT

seller\_id,

total\_price,

DENSE\_RANK() OVER (ORDER BY total\_price DESC) ranking

FROM

(

SELECT

s.seller\_id,

SUM(s.quantity\*p.unit\_price) AS total\_price

FROM

sales s

INNER JOIN

product p

ON

p.product\_id = s.product\_id

GROUP BY

seller\_id

) temp

)

SELECT

seller\_id

FROM

temp\_sales

WHERE

ranking = 1;

-- Q.66 Write an SQL Query that reports the buyers who have bought S8 but not iphone. Note that S8 and iphone

-- are products present in the product table. Return the result table in any order.

-- Same input table as for previous question i.e. 65

SELECT

s.buyer\_id

FROM

sales s

INNER JOIN

product p

ON

p.product\_id = s.product\_id

WHERE

p.product\_name = 'S8'

AND

s.buyer\_id NOT IN (

SELECT

s.buyer\_id

FROM

sales S

INNER JOIN

product P

ON

s.product\_id = p.product\_id

WHERE

p.product\_name = 'Iphone'

);

-- Q.67 Write an SQL Query to compute the moving average of how much the customer paid in a seven days window

-- (i.e., current day + 6 days before). average\_amount should be rounded to two decimal places.

-- Return result table ordered by visited\_on in ascending order.

CREATE TABLE customer(

customer\_id INT,

name VARCHAR(20),

visited\_on DATE,

amount INT,

CONSTRAINT PRIMARY\_KEY PRIMARY KEY(customer\_id,visited\_on)

);

INSERT INTO customer VALUES

(1,'JOHN','2019-01-01',100),

(2,'DANIEL','2019-01-02',110),

(3,'JADE','2019-01-03',120),

(4,'KHALED','2019-01-04',130),

(5,'WINSTON','2019-01-05',110),

(6,'ELVIS','2019-01-06',140),

(7,'ANNA','2019-01-07',150),

(8,'MARIA','2019-01-08',80),

(9,'JAZE','2019-01-09',110),

(1,'JOHN','2019-01-10',130),

(3,'JADE','2019-01-10',150);

WITH temp\_customer AS (

SELECT

visited\_on,

SUM(amount) AS amount

FROM

customer

GROUP BY

visited\_on

),

temp\_customer2 AS (

SELECT

visited\_on,

SUM(amount) OVER(ORDER BY visited\_on ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS weekly\_amount,

ROUND(AVG(amount) OVER(ORDER BY visited\_on ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 2) AS average\_amount,

DENSE\_RANK() OVER(ORDER BY visited\_on) as ranking

FROM

temp\_customer

)

SELECT

visited\_on,

weekly\_amount,

average\_amount

FROM

temp\_customer2

WHERE

ranking > 6;

-- Q.68 Write an SQL Query to find the total score for each gender on each day.

-- Return the result table ordered by gender and day in ascending order.

CREATE TABLE scores(

player\_name VARCHAR(20),

gender VARCHAR(20),

day DATE,

score\_points INT,

CONSTRAINT prime\_key PRIMARY KEY(gender,day)

);

INSERT INTO scores VALUES

('ARON','F','2020-01-01',17),

('ALICE','F','2020-01-07',23),

('BAJRANG','M','2020-01-07',7),

('KHALI','M','2019-12-25',11),

('SLAMAN','M','2019-12-30',13),

('JOE','M','2019-12-31',3),

('JOSE','M','2019-12-18',2),

('PRIYA','F','2019-12-31',23),

('PRIYANKA','F','2019-12-30',17);

SELECT

gender,

day,

SUM(score\_points) OVER(PARTITION BY gender ORDER BY day

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total\_points

FROM

scores

order by

gender,

day;

-- Q.69 Write an SQL Query to find the start and end number of continuous ranges in the table logs.

-- Return the result table ordered by start\_id.

CREATE TABLE logs(

log\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(log\_id)

);

INSERT INTO logs VALUES

(1),

(2),

(3),

(7),

(8),

(10);

SELECT

MIN(log\_id) AS start\_id,

MAX(log\_id) AS end\_id

FROM

(

SELECT

log\_id,

DENSE\_RANK() OVER(ORDER BY log\_id - RN) AS ranking

FROM

(

SELECT

log\_id,

ROW\_number() OVER(ORDER BY log\_id) AS RN

FROM

logs

) temp\_log

) temp\_log2

GROUP BY

ranking

ORDER BY

start\_id;

-- Q.70 Write an SQL Query to find the number of times each student attended each exam.

-- Return the result table ordered by student\_id and subject\_name.

CREATE TABLE students(

student\_id INT,

student\_name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(student\_id)

);

CREATE TABLE subjects(

subject\_name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(subject\_name)

);

CREATE TABLE exams(

student\_id INT,

subject\_name VARCHAR(20)

);

INSERT INTO students VALUES

(1,'ALICE'),

(2,'BOB'),

(13,'JOHN'),

(6,'ALEX');

INSERT INTO subjects VALUES

('MATHS'),

('PHYSICS'),

('PROGRAMMING');

INSERT INTO exams VALUES

(1,'MATHS'),

(1,'PHYSICS'),

(1,'PROGRAMMING'),

(2,'PROGRAMMING'),

(1,'PHYSICS'),

(1,'MATHS'),

(13,'MATHS'),

(13,'PROGRAMMING'),

(13,'PHYSICS'),

(2,'MATHS'),

(1,'MATHS');

WITH temp\_student AS (

SELECT

student\_id,

student\_name,

subject\_name

FROM

students,

subjects

),

temp\_student2 AS (

SELECT

student\_id,

subject\_name,

COUNT(\*) AS times\_attended\_each\_exam

FROM

exams

GROUP BY

student\_id,

subject\_name

)

SELECT

t.student\_id,

t.student\_name,

t.subject\_name,

COALESCE(times\_attended\_each\_exam,0) AS attended\_exams

FROM

temp\_student t

LEFT JOIN

temp\_student2 t2

ON

t.student\_id = t2.student\_id

AND

t.subject\_name = t2.subject\_name

ORDER BY

t.student\_id,

t.subject\_name;

-- Q.71 Write an SQL Query to find employee\_id of all employees that directly or indirectly

-- report their work to the head of the company. The indirect relation between managers will not exceed

-- three managers as the company is small. Return the result table in any order.

CREATE TABLE employees(

employee\_id INT,

employee\_name VARCHAR(20),

manager\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(employee\_id)

);

INSERT INTO employees VALUES

(1,'BOSS',1),

(3,'ALICE',3),

(2,'BOB',1),

(4,'DANIEL',2),

(7,'LUIS',4),

(8,'JHON',3),

(9,'ANGELA',8),

(77,'ROBERT',1);

with recursive managers as (

SELECT

employee\_id,

manager\_id

FROM

employees

WHERE

employee\_id = 1

UNION

SELECT

e.employee\_id,

m.manager\_id

FROM

managers m

INNER JOIN

employees e

ON

e.manager\_id = m.employee\_id

)

SELECT

employee\_id

FROM

managers

WHERE

employee\_id <> manager\_id;

-- Q.72 Write an SQL Query to find for each month and country, the number of transactions and their total amount,

-- the number of approved transactions and their total amount. Return the result table in any order.

CREATE TABLE transactions(

id INT,

country VARCHAR(20),

state ENUM ('APPROVED','DECLINED'),

amount INT,

trans\_date DATE,

CONSTRAINT prime\_key PRIMARY KEY(id)

);

INSERT INTO transactions VALUES

(121,'US','APPROVED',1000,'2018-12-18'),

(122,'US','DECLINED',2000,'2018-12-19'),

(123,'US','APPROVED',2000,'2019-01-01'),

(124,'DE','APPROVED',2000,'2019-01-07');

WITH temp\_transactions AS (

SELECT

concat(YEAR(trans\_date), '-',MONTH(trans\_date)) AS transaction\_date,

country,

state,

count(\*) OVER (PARTITION BY concat(YEAR(trans\_date), '-',MONTH(trans\_date)), country) AS total\_transactions,

sum(amount) OVER (PARTITION BY concat(YEAR(trans\_date), '-',MONTH(trans\_date)), country) AS total\_transactions\_amount,

sum(amount) OVER (PARTITION BY concat(YEAR(trans\_date), '-',MONTH(trans\_date)), country, state) AS amount

FROM

transactions

)

SELECT

transaction\_date,

country,

total\_transactions,

count(\*) OVER(PARTITION BY transaction\_date, country, state) AS approved\_transactions,

total\_transactions\_amount,

amount AS approved\_amount

FROM

temp\_transactions

WHERE

state = 'Approved';

-- Q.73 Write an SQL Query to find the average daily percentage of posts that got

-- removed after being reported as spam, rounded to 2 decimal places.

CREATE TABLE actions(

user\_id INT,

post\_id INT,

action\_date DATE,

action ENUM ('VIEW','LIKE','REACTION','COMMENT','REPORT','SHARE'),

extra VARCHAR(20)

);

CREATE TABLE removals(

post\_id INT,

remove\_date DATE,

CONSTRAINT prime\_key PRIMARY KEY(post\_id)

);

INSERT INTO actions VALUES

(1,1,'2019-07-01','VIEW','NULL'),

(1,1,'2019-07-01','LIKE','NULL'),

(1,1,'2019-07-01','SHARE','NULL'),

(2,2,'2019-07-04','VIEW','NULL'),

(2,2,'2019-07-04','REPORT','SPAM'),

(3,4,'2019-07-04','VIEW','NULL'),

(3,4,'2019-07-04','REPORT','SPAM'),

(4,3,'2019-07-02','VIEW','NULL'),

(4,3,'2019-07-02','REPORT','SPAM'),

(5,2,'2019-07-03','VIEW','NULL'),

(5,2,'2019-07-03','REPORT','RACISM'),

(5,5,'2019-07-03','VIEW','NULL'),

(5,5,'2019-07-03','REPORT','RACISM');

INSERT INTO removals VALUES

(2,'2019-07-20'),

(3,'2019-07-18');

WITH temp\_action AS (

SELECT

action\_date,

post\_id,

COUNT(EXTRA) OVER(PARTITION BY action\_date) num\_post\_reported\_spam

FROM

actions

WHERE

extra = 'SPAM'

)

SELECT

ROUND(AVG(percentage), 2) AS avg\_daily\_percent

FROM

(

SELECT

action\_date,

ROUND((COUNT(post\_id)/num\_post\_reported\_spam) \* 100, 2) AS percentage

FROM

temp\_action

WHERE

post\_id IN (

SELECT

post\_id

FROM

removals

)

GROUP BY

action\_date

) temp;

-- Q.74 SAME AS Q.43

-- Q.75 SAME AS Q.43

-- Q.76 Write an SQL Query to find the salaries of the employees after applying taxes.

-- Round the salary to the nearest integer.

CREATE TABLE salaries(

company\_id INT,

employee\_id INT,

employee\_name VARCHAR(20),

salary INT,

CONSTRAINT prime\_key PRIMARY KEY(company\_id, employee\_id)

);

INSERT INTO salaries VALUES

(1,1,'TONY',2000),

(1,2,'PRONUB',21300),

(1,3,'TYRROX',10800),

(2,1,'PAM',300),

(2,7,'BASSEM',450),

(2,9,'HERMIONE',700),

(3,7,'BOCABEN',100),

(3,2,'OGNJEN',2200),

(3,13,'NYAN CAT',3300),

(3,15,'MORNING CAT',7777);

WITH temp\_salaries AS (

SELECT

company\_id,

employee\_id,

employee\_name,

salary,

MAX(salary) OVER(PARTITION BY company\_id) max\_sal\_per\_company

FROM

salaries

)

SELECT

company\_id,

employee\_id,

employee\_name,

salary,

ROUND(

CASE

WHEN max\_sal\_per\_company > 10000 THEN salary - (salary \* 0.49)

WHEN max\_sal\_per\_company BETWEEN 1000 AND 10000 THEN salary - (salary \* 0.24)

ELSE salary

END, 0) AS sal\_after\_tax\_deduction

FROM

temp\_salaries;

-- Q.77 Write an SQL Query to evaluate the boolean expressions in Expressions table.

-- Return the result table in any order.

CREATE TABLE variables(

name VARCHAR(2),

value INT,

CONSTRAINT prime\_key PRIMARY KEY(name)

);

INSERT INTO variables VALUES

('x',66),

('y',77);

CREATE TABLE expressions(

left\_operand VARCHAR(2),

operator ENUM('<','=','>'),

right\_operand VARCHAR(2),

CONSTRAINT prime\_key PRIMARY KEY(left\_operand, operator, right\_operand)

);

INSERT INTO expressions VALUES

('x','>','y'),

('x','<','y'),

('x','=','y'),

('y','>','x'),

('y','<','x'),

('x','=','x');

SELECT

e.left\_operand,

e.operator,

e.right\_operand,

CASE

WHEN e.operator = '<' THEN IF(l.value < r.value, 'TRUE', 'FALSE')

WHEN e.operator = '>' THEN IF(l.value > r.value, 'TRUE', 'FALSE')

ELSE IF(l.value = r.value, 'TRUE', 'FALSE')

END AS result

FROM

expressions e

JOIN

variables l

ON

e.left\_operand = l.name

JOIN

variables r

ON

e.right\_operand = r.name;

-- Q.78 SAME AS Q.55

-- Q.79 Write a Query that prints a list of employee names (i.e.: the name attribute)

-- from the employee table in alphabetical order.

CREATE TABLE employee(

employee\_id INT,

name VARCHAR(20),

months INT,

salary INT

);

INSERT INTO employee VALUES

(12228,'ROSE',15,1968),

(33645,'ANGELA',1,3443),

(45692,'FRANK',17,1608),

(56118,'PATRIK',7,1345),

(74197,'KINBERLY',16,4372),

(78454,'BONNIE',8,1771),

(83565,'MICHAEL',6,2017),

(98607,'TODD',5,3396),

(99989,'JOE',9,3573);

SELECT

name

FROM

employee

ORDER BY

name;

-- Q.80 Write a Query to obtain the year-on-year growth rate for the total spend of each product for each year.

CREATE TABLE user\_transactions(

transaction\_id INT,

product\_id INT,

spend FLOAT,

transaction\_date VARCHAR(30)

);

INSERT INTO user\_transactions VALUES

(1341,123424,1500.60,'12/31/2019 12:00:00'),

(1423,123424,1000.20,'12/31/2020 12:00:00'),

(1623,123424,1246.44,'12/31/2021 12:00:00'),

(1322,123424,2145.32,'12/31/2022 12:00:00');

-- Approach 1

WITH temp\_transactions AS (

SELECT

product\_id,

transaction\_date,

spend AS curr\_year\_spend,

LAG(spend,1,0) OVER w AS prev\_year\_spend,

IFNULL(spend - LAG(spend,1) OVER w, 0) AS prev\_curr\_spend\_diff

FROM

user\_transactions

WINDOW

w AS (PARTITION BY product\_id ORDER BY EXTRACT(YEAR FROM transaction\_date))

)

SELECT

product\_id,

curr\_year\_spend,

ROUND(prev\_year\_spend, 2),

IFNULL(ROUND((prev\_curr\_spend\_diff \* 100)/prev\_year\_spend,2),0) AS YOY

FROM

temp\_transactions;

-- Approach 2

WITH temp\_transactions AS (

SELECT

YEAR(STR\_TO\_date(transaction\_date, '%m/%d/%Y')) AS YEAR\_id,

product\_id,

transaction\_date,

spend AS curr\_year\_spend,

LAG(spend,1) OVER w AS prev\_year\_spend,

spend - LAG(spend,1) OVER w AS prev\_curr\_spend\_diff

FROM

user\_transactions

WINDOW

w AS (PARTITION BY product\_id ORDER BY EXTRACT(YEAR FROM transaction\_date))

)

SELECT

year\_id,

product\_id,

curr\_year\_spend,

ROUND(prev\_year\_spend, 2),

ROUND((prev\_curr\_spend\_diff \* 100)/prev\_year\_spend,2) AS YOY

FROM

temp\_transactions;

-- Approach 3

SELECT

product\_id,

YEAR(STR\_TO\_date(transaction\_date, '%m/%d/%Y')) AS YEAR\_id,

spend AS curr\_year\_spend,

ROUND(LAG(spend,1,0) OVER w ,2) AS prev\_year\_spend,

ROUND((spend - LAG(spend,1) OVER w ) \* 100 /

LAG(spend,1) OVER w, 2) AS YOY

FROM

user\_transactions

WINDOW

w AS (PARTITION BY product\_id ORDER BY EXTRACT(YEAR FROM transaction\_date));

-- Q.81 Write a SQL Query to find the number of prime and non-prime items that can be stored

-- in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.

CREATE TABLE inventory(

item\_id INT,

item\_type VARCHAR(20),

item\_category VARCHAR(20),

square\_foot FLOAT

);

INSERT INTO inventory VALUES

(1374,'PRIME\_ELIGIBLE','MINI FRidGE',68.00),

(4245,'NOT\_PRIME','STANDING LAMP',26.40),

(2452,'PRIME\_ELIGIBLE','TELEVISION',85.00),

(3255,'NOT\_PRIME','SidE TABLE',22.60),

(1672,'PRIME\_ELIGIBLE','LAPTOP',8.50);

-- Approach 1

WITH temp\_inventory AS (

SELECT

item\_type,

SUM(square\_foot) AS square\_foot\_per\_category,

COUNT(\*) AS count\_of\_items

FROM

inventory

GROUP BY

item\_type

),

temp\_inventory2 AS (

SELECT

(500000 - SUM(square\_foot\_per\_category)\*FLOOR(500000/SUM(square\_foot\_per\_category))) AS area\_left

FROM

temp\_inventory

WHERE

item\_type = 'PRIME\_ELIGIBLE'

),

temp\_inventory3 AS (

SELECT

item\_type,

CASE

WHEN item\_type = 'PRIME\_ELIGIBLE'

THEN FLOOR(500000/square\_foot\_per\_category) \* count\_of\_items

WHEN item\_type = 'NOT\_PRIME'

THEN FLOOR((SELECT area\_left FROM temp\_inventory2) / square\_foot\_per\_category) \* count\_of\_items

END AS item\_count

FROM

temp\_inventory

)

SELECT

item\_type,

item\_count

FROM

temp\_inventory3;

-- Approach 2

WITH temp\_inventory AS (

SELECT

item\_type,

SUM(square\_foot) AS square\_foot\_per\_category,

COUNT(\*) AS count\_of\_items,

CASE

WHEN item\_type = 'PRIME\_ELIGIBLE' THEN FLOOR(500000/SUM(square\_foot)) \* COUNT(\*)

END AS prime\_items\_count

FROM

inventory

GROUP BY

item\_type

),

temp\_inventory2 AS (

SELECT

(500000 - SUM(square\_foot\_per\_category)\*FLOOR(500000/SUM(square\_foot\_per\_category))) AS area\_left

FROM

temp\_inventory

WHERE

item\_type = 'PRIME\_ELIGIBLE'

),

temp\_inventory3 AS (

SELECT

item\_type,

CASE

WHEN item\_type = 'PRIME\_ELIGIBLE'

THEN prime\_items\_count

WHEN item\_type = 'NOT\_PRIME'

THEN FLOOR((SELECT area\_left FROM temp\_inventory2) / square\_foot\_per\_category) \* count\_of\_items

END AS item\_count

FROM

temp\_inventory

)

SELECT

item\_type,

item\_count

FROM

temp\_inventory3;

-- Approach 3

WITH temp\_inventory AS (

SELECT

item\_type,

SUM(square\_foot) AS square\_foot\_per\_category,

COUNT(\*) AS count\_of\_items,

CASE

WHEN item\_type = 'PRIME\_ELIGIBLE' THEN FLOOR(500000/SUM(square\_foot)) \* COUNT(\*)

END AS prime\_items\_count

FROM

inventory

GROUP BY

item\_type

),

temp\_inventory2 AS (

SELECT

item\_type,

CASE

WHEN item\_type = 'PRIME\_ELIGIBLE' THEN prime\_items\_count

WHEN item\_type = 'NOT\_PRIME' THEN FLOOR((500000 - (

SELECT

SUM(square\_foot\_per\_category)

FROM

temp\_inventory

WHERE

item\_type = 'PRIME\_ELIGIBLE'

) \* FLOOR(500000 /

(

SELECT

SUM(square\_foot\_per\_category)

FROM

temp\_inventory

WHERE

item\_type = 'PRIME\_ELIGIBLE'

))) / square\_foot\_per\_category) \* count\_of\_items

END AS item\_count

FROM

temp\_inventory

)

SELECT

item\_type,

item\_count

FROM

temp\_inventory2;

-- Q.82 Write a Query to obtain the active user retention in July 2022.

-- Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

CREATE TABLE user\_actions(

user\_id INT,

event\_id INT,

event\_type ENUM('SIGN-IN','LIKE','COMMENT'),

event\_date DATETIME

);

INSERT INTO user\_actions VALUES

(445,7765,'SIGN-IN','2022-05-31 12:00:00'),

(742,6458,'SIGN-IN','2022-06-03 12:00:00'),

(445,3634,'LIKE','2022-06-05 12:00:00'),

(742,1374,'COMMENT','2022-06-05 12:00:00'),

(648,3124,'LIKE','2022-06-18 12:00:00');

-- Approach 1

WITH temp\_actions AS (

SELECT

user\_id,

event\_date,

event\_type,

SUBSTR(event\_date, 6, 2) - lag(SUBSTR(event\_date, 6, 2)) OVER w AS difference

FROM

user\_actions

WINDOW

w as (PARTITION BY user\_id ORDER BY event\_date)

),

temp\_actions2 AS (

SELECT

SUBSTR(event\_date, 6, 2) AS months,

COUNT(user\_id) AS monthly\_active\_users

FROM

temp\_actions

WHERE

difference = 1 AND event\_type IN ('LIKE', 'COMMENT', 'SIGN-IN')

GROUP BY

months

)

SELECT

months,

monthly\_active\_users

FROM

temp\_actions2;

-- Approach 2

WITH temp\_actions AS (

SELECT

user\_id,

event\_date,

event\_type,

SUBSTR(event\_date, 6, 2) - lag(SUBSTR(event\_date, 6, 2)) OVER w AS difference

FROM

user\_actions

WINDOW

w as (PARTITION BY user\_id ORDER BY event\_date)

)

SELECT

SUBSTR(event\_date, 6, 2) AS months,

COUNT(DISTINCT user\_id) AS active\_users

FROM

temp\_actions

WHERE

difference = 1

AND

event\_type IN ('LIKE', 'COMMENT', 'SIGN-IN')

GROUP BY

months;

-- Q.83 Write a Query to report the median of searches made by a user.

-- Round the median to one decimal point.

CREATE TABLE search\_frequency(

searches INT,

num\_users INT

);

INSERT INTO search\_frequency VALUES

(1,2),

(2,2),

(3,3),

(4,1);

WITH temp\_search\_freq AS (

SELECT

searches,

num\_users,

ROW\_NUMBER() OVER(ORDER BY searches) row\_num,

COUNT(\*) OVER(ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) total\_records

FROM

search\_frequency

),

temp\_search\_freq2 as (

SELECT

searches,

num\_users,

CASE

WHEN total\_records % 2 <> 0 THEN (

SELECT

DISTINCT ROUND(SUM(searches) OVER w /

COUNT(\*) OVER w,1)

FROM

temp\_search\_freq

WHERE

row\_num = ROUND((total\_records + 1) / 2, 0)

WINDOW

w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

)

WHEN total\_records % 2 = 0 THEN (

SELECT

DISTINCT ROUND(SUM(searches) OVER w /

COUNT(\*) OVER w,1)

FROM

temp\_search\_freq

WHERE

row\_num IN (total\_records/2,(total\_records/2)+1)

WINDOW

w AS (ORDER BY searches ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

)

END AS median

FROM

temp\_search\_freq

)

SELECT

DISTINCT median

FROM

temp\_search\_freq2;

-- Q.84 Write a Query to update the Facebook advertisers status using the daily\_pay table.

-- Advertiser is a two-column table containing the user id and their payment status based

-- on the last payment and daily\_pay table has current information about their payment.

-- Only advertisers who paid will show up in this table.

-- Output the user id and current payment status sorted by the user id.

CREATE TABLE advertiser(

user\_id VARCHAR(20),

status ENUM('NEW','EXISTING','CHURN','RESURRECT')

);

CREATE TABLE daily\_pay(

user\_id VARCHAR(20),

paid DECIMAL

);

INSERT INTO advertiser VALUES

('BING','NEW'),

('YAHOO','NEW'),

('ALIBABA','EXISTING');

INSERT INTO daily\_pay VALUES

('YAHOO',45.00),

('ALIBABA',100.00),

('TARGET',13.00);

SELECT

user\_id,

CASE

WHEN user\_id IN (SELECT user\_id FROM daily\_pay) THEN 'EXISTING'

ELSE 'CHURN'

END AS new\_status

FROM

advertiser

ORDER BY

user\_id;

-- Q.85 Write a SQL Query that calculates the total time that the fleet of

-- servers was running. The output should be in units of full days.

CREATE TABLE server\_utilization(

server\_id INT,

session\_status VARCHAR(20),

status\_time VARCHAR(25)

);

INSERT INTO server\_utilization VALUES

(1,'start','08/02/2022 10:00:00'),

(1,'stop','08/04/2022 10:00:00'),

(2,'stop','08/24/2022 10:00:00'),

(2,'start','08/17/2022 10:00:00');

-- Approach 1

SELECT

stop\_time - start\_time AS total\_up\_time

FROM

(

SELECT

SUM(

CASE

WHEN session\_status = 'start' then EXTRACT(DAY from STR\_TO\_DATE(status\_time, '%m/%d/%y'))

END

) AS start\_time,

SUM(

CASE

WHEN session\_status = 'stop' then EXTRACT(DAY from STR\_TO\_DATE(status\_time, '%m/%d/%y'))

END

) AS stop\_time

FROM

server\_utilization

) temp\_server\_utilization;

-- Approach 2 (Works only in PostgreSQL)

WITH temp\_server\_utilization AS (

SELECT

server\_id,

status\_time AS start\_time,

session\_status,

lead(status\_time) OVER(ORDER BY server\_id,status\_time) AS end\_time

FROM

server\_utilization

)

SELECT

EXTRACT(DAY FROM justify\_hours(SUM(end\_time - start\_time))) as total\_time

FROM

temp\_server\_utilization

WHERE

session\_status = 'start';

-- Q.86 Sometimes, payment transactions are repeated by accident; it could be due to user error,

-- API failure or a retry error that causes a credit card to be charged twice.

-- Using the transactions table, identify any payments made at the same merchant with the

-- same credit card for the same amount within 10 minutes of each other. Count such repeated payments.

CREATE TABLE transactions(

transaction\_id INT,

merchant\_id INT,

credit\_card\_id INT,

amount INT,

transaction\_timestamp DATETIME

);

INSERT INTO transactions VALUES

(1,101,1,100,'2022-09-25 12:00:00'),

(2,101,1,100,'2022-09-25 12:08:00'),

(3,101,1,100,'2022-09-25 12:28:00'),

(4,102,2,300,'2022-09-25 12:00:00'),

(5,102,2,400,'2022-09-25 14:00:00');

WITH temp\_transactions AS (

SELECT

merchant\_id,

credit\_card\_id,

amount,

transaction\_timestamp,

LAG(transaction\_timestamp) OVER w AS prev\_tran\_timestamp,

timestampdiff(MINUTE,LAG(transaction\_timestamp) OVER w, transaction\_timestamp) AS difference

FROM

transactions

WINDOW

w as (PARTITION BY credit\_card\_id ORDER BY transaction\_timestamp)

)

SELECT

COUNT(DISTINCT merchant\_id) AS payment\_count

FROM

temp\_transactions

WHERE

difference <= 10;

-- Approach 2 (Works only in PostgreSQL)

WITH temp\_transactions AS (

SELECT

merchant\_id,

credit\_card\_id,

amount,

transaction\_timestamp,

LAG(transaction\_timestamp) OVER w AS prev\_tran\_timestamp,

EXTRACT(EPOCH FROM LAG(transaction\_timestamp) OVER w - transaction\_timestamp) AS difference

FROM

transactions

WINDOW

w as (PARTITION BY credit\_card\_id ORDER BY transaction\_timestamp)

)

SELECT

COUNT(DISTINCT merchant\_id) AS payment\_count

FROM

temp\_transactions

WHERE

difference < 10;

-- Q.87 Write a SQL Query to find the bad experience rate in the first 14 days for new users who signed

-- up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.

CREATE TABLE orders(

order\_id INT,

customer\_id INT,

trip\_id INT,

status ENUM('COMPLETED SUCCESSFULLY','COMPLETED INCORRECTLY','NEVER\_RECEIVED'),

order\_timestamp VARCHAR(30)

);

INSERT INTO orders VALUES

(727424,8472,100463,'COMPLETED SUCCESSFULLY','06/05/2022 09:12:00'),

(242513,2341,100482,'COMPLETED INCORRECTLY','06/05/2022 14:40:00'),

(141367,1314,100362,'COMPLETED INCORRECTLY','06/07/2022 15:03:00'),

(582193,5421,100657,'NEVER\_RECEIVED','07/07/2022 15:22:00'),

(253613,1314,100213,'COMPLETED SUCCESSFULLY','06/12/2022 13:43:00');

CREATE TABLE trips(

dasher\_id INT,

trip\_id INT,

estimated\_delivery\_timestamp VARCHAR(25),

actual\_delivery\_timestamp VARCHAR(25)

);

INSERT INTO TRIPS VALUES

(101,100463,'06/05/2022 09:42:00','06/05/2022 09:38:00'),

(102,100482,'06/05/2022 15:10:00','06/05/2022 15:46:00'),

(101,100362,'06/07/2022 15:33:00','06/07/2022 16:45:00'),

(102,100657,'07/07/2022 15:52:00',NULL),

(103,100213,'06/12/2022 14:13:00','06/12/2022 14:10:00');

CREATE TABLE customers(

customer\_id INT,

signup\_timestamp VARCHAR(30)

);

INSERT INTO customers VALUES

(8472,'05/30/2022 00:00:00'),

(2341,'06/01/2022 00:00:00'),

(1314,'06/03/2022 00:00:00'),

(1435,'06/05/2022 00:00:00'),

(5421,'06/07/2022 00:00:00');

SELECT

ROUND(

SUM(

CASE

WHEN status !='completed successfully' THEN 1 ELSE 0

END

)\*100.0/count(\*),2) AS bad\_experience\_pct

FROM

customers C

INNER JOIN

orders O

ON

o.customer\_id = c.customer\_id

WHERE

o.order\_timestamp < date\_add(STR\_TO\_date(signup\_timestamp, '%m/%d/%Y'), INTERVAL 14 DAY)

AND

MONTH(STR\_TO\_date(signup\_timestamp, '%m/%d/%Y')) = 06

AND

YEAR(STR\_TO\_date(signup\_timestamp, '%m/%d/%Y')) = 2022;

-- Q.88 SAME AS 68

-- Q.89 SAME AS 55

-- Q.90 Write an SQL Query to report the median of all the numbers in the database

-- after decompressing the numbers table. Round the median to one decimal point.

CREATE TABLE numbers(

num INT,

frequency INT

);

INSERT INTO numbers VALUES

(0,7),

(1,1),

(2,3),

(3,1);

WITH RECURSIVE num\_frequency (num,frequency, i) AS

(

SELECT

num,

frequency,1

FROM

numbers

UNION ALL

SELECT

num,

frequency,

i+1

FROM

num\_frequency

WHERE

num\_frequency.i < num\_frequency.frequency

),

num\_frequency2 AS (

SELECT

num,

frequency,

row\_number() OVER(ORDER BY num, frequency) AS row\_num,

COUNT(\*) OVER(ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS total\_records

FROM

num\_frequency

)

SELECT

DISTINCT CASE

WHEN total\_records % 2 <> 0 THEN (

SELECT

DISTINCT ROUND(SUM(num) OVER w /

COUNT(\*) OVER w, 1)

FROM

num\_frequency2

WHERE

row\_num = ROUND((total\_records + 1) / 2, 0)

WINDOW

w as (ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))

WHEN total\_records % 2 = 0 THEN (

SELECT

DISTINCT ROUND(SUM(num) OVER w /

COUNT(\*) OVER w, 1)

FROM

num\_frequency2

WHERE

row\_num IN (total\_records/2,(total\_records/2)+1)

WINDOW

w as (ORDER BY num, frequency ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING))

END AS median

FROM

num\_frequency2;

-- Q.91 Write an SQL Query to report the comparison result (higher/lower/same) of the average salary of

-- employees in a department to the companys average salary. Return the result table in any order.

CREATE TABLE salary(

id INT,

employee\_id INT,

amount INT,

paydate DATE,

CONSTRAINT prime\_key PRIMARY KEY(id)

);

CREATE TABLE employee(

employee\_id INT,

department\_id INT,

CONSTRAINT prime\_key PRIMARY KEY(employee\_id)

);

INSERT INTO salary VALUES

(1,1,9000,'2017/03/31'),

(2,2,6000,'2017/03/31'),

(3,3,10000,'2017/03/31'),

(4,1,7000,'2017/02/28'),

(5,2,6000,'2017/02/28'),

(6,3,8000,'2017/02/28');

INSERT INTO employee VALUES

(1,1),

(2,2),

(3,2);

WITH temp\_comparison AS (

SELECT

s.employee\_id,

e.department\_id,

s.amount,

s.paydate,

avg(amount) OVER (PARTITION BY MONTH(paydate) ORDER BY month(paydate), employee\_id

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) company\_avg\_salary,

avg(amount) OVER (PARTITION BY MONTH(paydate), department\_id order by month(paydate)

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) department\_avg

FROM

salary s

INNER JOIN

employee e

ON

e.employee\_id = s.employee\_id

)

SELECT

DISTINCT DATE\_FORMAT(paydate, '%Y-%m') AS pay\_month,

department\_id,

CASE

WHEN company\_avg\_salary = department\_avg THEN 'same'

WHEN company\_avg\_salary > department\_avg THEN 'lower'

WHEN company\_avg\_salary < department\_avg THEN 'higher'

END AS comparison

FROM

temp\_comparison;

-- Q.92 Write an SQL Query to report for each install date, the number of players

-- that installed the game on that day, and the day one retention.

CREATE TABLE activity(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT,

CONSTRAINT prime\_key PRIMARY KEY(player\_id, event\_date)

);

INSERT INTO activity VALUES

(1,2,'2016-03-01',5),

(1,2,'2016-03-02',6),

(2,3,'2017-06-25',1),

(3,1,'2016-03-01',0),

(3,4,'2018-07-03',5);

-- Approach 1

SELECT

a.event\_date AS install\_date,

COUNT(a.player\_id) AS installs,

ROUND(COUNT(b.player\_id) / COUNT(a.player\_id), 2) AS day1\_retention

FROM

(

SELECT

player\_id,

MIN(event\_date) AS event\_date

FROM

activity

GROUP BY

player\_id

) a

LEFT JOIN

activity b

ON

a.player\_id = b.player\_id

AND

a.event\_date + 1 = b.event\_date

GROUP BY

a.event\_date;

-- Approach 2

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;

-- Q.93 SAME AS 50

-- Q.94 Write an SQL Query to report the students (student\_id, student\_name) being -- Quiet in all exams.

-- Do not return the student who has never taken any exam.

CREATE TABLE student(

student\_id INT,

student\_name VARCHAR(20),

CONSTRAINT prime\_key PRIMARY KEY(student\_id)

);

CREATE TABLE exam(

exam\_id INT,

student\_id INT,

score INT,

CONSTRAINT prime\_key PRIMARY KEY(exam\_id,student\_id)

);

INSERT INTO student VALUES

(1,'DANIEL'),

(2,'JADE'),

(3,'STELLA'),

(4,'JONATHAN'),

(5,'WILL');

INSERT INTO exam VALUES

(10,1,70),

(10,2,80),

(10,3,90),

(20,1,80),

(30,1,70),

(30,3,80),

(30,4,90),

(40,1,60),

(40,2,70),

(40,4,80);

-- Approach 1

WITH temp\_examination AS (

SELECT

exam\_id,

student\_id,

score,

max(score) OVER w AS highest,

min(score) OVER w AS lowest

FROM

exam

WINDOW

w AS (PARTITION BY exam\_id)

),

temp\_examination1 AS (

SELECT

DISTINCT student\_id

FROM

temp\_examination

WHERE

score IN (lowest, highest)

)

SELECT

DISTINCT s.student\_id,

s.student\_name

FROM

temp\_examination

INNER JOIN

student s

ON

s.student\_id = temp\_examination.student\_id

WHERE

s.student\_id NOT IN (SELECT student\_id FROM temp\_examination1);

-- Approach 2

WITH temp\_examination AS (

SELECT

student\_id,

CASE

WHEN score < max(score) OVER(PARTITION BY exam\_id) AND score >min(score) OVER(PARTITION BY exam\_id) THEN 0

ELSE 1

END AS category

FROM

exam

ORDER BY

student\_id

),

temp\_examination1 AS (

SELECT

student\_id,

SUM(category) AS high\_low\_count

FROM

temp\_examination

GROUP BY

student\_id

)

SELECT

s.student\_id,

s.student\_name

FROM

student s

INNER JOIN

temp\_examination1

ON

s.student\_id = temp\_examination1.student\_id

WHERE

high\_low\_count = 0

ORDER BY

temp\_examination1.student\_id;

-- Q.95 SAME AS 94

-- Q.96 Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022

-- sorted in descending order.

CREATE TABLE songs\_history(

history\_id INT,

user\_id INT,

song\_id INT,

song\_plays INT

);

INSERT INTO songs\_history VALUES

(10011,777,1238,11),

(12452,695,4520,1);

CREATE TABLE songs\_weekly(

user\_id INT,

song\_id INT,

listen\_time VARCHAR(25)

);

INSERT INTO songs\_weekly VALUES

(777,1238,'08/01/2022 12:00:00'),

(695,4520,'08/04/2022 08:00:00'),

(125,9630,'08/04/2022 16:00:00'),

(695,9852,'08/07/2022 12:00:00');

WITH streaming AS (

SELECT

user\_id,

song\_id,

song\_plays

FROM

songs\_history

UNION ALL

SELECT

user\_id,

song\_id,

count(\*) AS song\_plays

FROM

songs\_weekly

WHERE

listen\_time <= '08/04/2022 23:59:59'

GROUP by

user\_id,

song\_id

)

SELECT

user\_id,

song\_id,

SUM(song\_plays) as song\_plays

FROM

streaming

GROUP BY

user\_id,

song\_id

ORDER BY

song\_plays DESC;

-- Q.97 Write a query to find the confirmation rate of users who confirmed their signups with text messages.

-- Round the result to 2 decimal places.

CREATE TABLE emails(

email\_id INT,

user\_id INT,

signup\_date DATETIME

);

INSERT INTO emails VALUES

(125,7771,'2022-06-14 00:00:00'),

(236,6950,'2022-07-01 00:00:00'),

(433,1052,'2022-07-09 00:00:00');

CREATE TABLE texts(

text\_id INT,

email\_id INT,

signup\_action VARCHAR(20)

);

INSERT INTO texts VALUES

(6878,125,'CONFIRMED'),

(6920,236,'NOT CONFIRMED'),

(6994,236,'CONFIRMED');

WITH temp\_confirmation AS (

SELECT

e.email\_id,

CASE

WHEN signup\_action = 'Confirmed' THEN 1

END

AS confirmed\_users

FROM

emails e

LEFT JOIN

texts t

ON

e.email\_id = t.email\_id

AND

t.signup\_action = 'Confirmed'

)

SELECT

ROUND(SUM(confirmed\_users)/COUNT(email\_id),2) AS confirm\_rate

FROM

temp\_confirmation;

-- Q.98 Calculate the 3-day rolling average of tweets published by each user for each date

-- that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.

CREATE TABLE tweets(

tweet\_id INT,

user\_id INT,

tweet\_date DATETIME

);

INSERT INTO TWEETS VALUES

(214252,111,'2022-06-01 12:00:00'),

(739252,111,'2022-06-01 12:00:00'),

(846402,111,'2022-06-02 12:00:00'),

(241425,254,'2022-06-02 12:00:00'),

(137374,111,'2022-06-04 12:00:00');

WITH temp\_tweets AS (

SELECT

user\_id,

tweet\_date,

COUNT(tweet\_id) AS tweets\_count

FROM

tweets

GROUP BY

user\_id,

tweet\_date

ORDER BY

user\_id,

tweet\_date

)

SELECT

user\_id,

tweet\_date,

ROUND(avg(tweets\_count)

OVER(PARTITION BY user\_id ORDER BY tweet\_date

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),2) AS rolling\_avg\_3days

FROM

temp\_tweets;

-- Q.99 Write a query to obtain a breakdown of the time spent sending vs. opening snaps

-- (as a percentage of total time spent on these activities) for each age group.

CREATE TABLE activities(

activity\_id INT,

user\_id INT,

activity\_type ENUM('SEND','OPEN','CHAT'),

time\_spent FLOAT,

activity\_date varchar(25)

);

INSERT INTO activities VALUES

(7274,123,'OPEN',4.50,'06/22/2022 12:00:00'),

(2425,123,'SEND',3.50,'06/22/2022 12:00:00'),

(1413,456,'SEND',5.67,'06/23/2022 12:00:00'),

(1414,789,'CHAT',11.00,'06/25/2022 12:00:00'),

(2536,456,'OPEN',3.00,'06/25/2022 12:00:00');

CREATE TABLE age\_breakdown(

user\_id INT,

age\_bucket ENUM('21-25','26-30','31-35')

);

INSERT INTO age\_breakdown VALUES

(123,'31-35'),

(456,'26-30'),

(789,'21-25');

WITH temp\_activities AS (

SELECT

user\_id,

activity\_type,

sum(time\_spent) time\_spent,

CASE

WHEN activity\_type = 'open' THEN sum(time\_spent)

ELSE 0

END opening\_snap,

CASE

WHEN activity\_type = 'send' THEN sum(time\_spent)

ELSE 0

END sending\_snap

FROM

activities

WHERE

activity\_type in ('open','send')

GROUP BY

user\_id,

activity\_type

ORDER BY

user\_id

),

temp\_activities2 AS (

SELECT

user\_id,

SUM(opening\_snap) time\_sending,

SUM(sending\_snap) time\_opening

FROM

temp\_activities

GROUP BY

user\_id

)

SELECT

ab.age\_bucket,

ROUND(time\_opening \* 100.0 /(time\_sending+time\_opening), 2) AS send\_perc,

ROUND(time\_sending \* 100.0 /(time\_sending+time\_opening), 2) AS open\_perc

FROM

temp\_activities2

INNER JOIN

age\_breakdown ab

ON

ab.user\_id = temp\_activities2.user\_id

ORDER BY

ab.age\_bucket;

-- Q.100 Write a query to return the IDs of these LinkedIn power creators in ascending order.

CREATE TABLE personal\_profiles(

profile\_id INT,

name VARCHAR(20),

followers INT

);

INSERT INTO personal\_profiles 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 INT,

company\_id INT

);

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 INT,

name VARCHAR(30),

followers INT

);

INSERT INTO company\_pages VALUES

(1,'THE DATA SCIENCE PODCAST',8000),

(2,'AIRBNB',700000),

(3,'THE RAVIT SHOW',6000),

(4,'DATA LEMUR',200),

(5,'YOUTUBE',16000000),

(6,'DATASCIENCE.VIN',4500),

(9,'ACE THE DATA SCIENCE INTERVIEW',4479);

SELECT

DISTINCT p.profile\_id

FROM

personal\_profiles p

INNER JOIN

employee\_company ec

ON

p.profile\_id = ec.personal\_profile\_id

INNER JOIN

company\_pages c

ON

ec.company\_id = c.company\_id

WHERE

p.followers > c.followers

ORDER BY

p.profile\_id;

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

-- If the user only hAS one activity, return that one. A user cannot perform more than one activity at the same time.

-- Return the result table in any order.

CREATE TABLE user\_activity(

username VARCHAR(20),

activity VARCHAR(20),

start\_date DATE,

end\_date DATE

);

INSERT INTO user\_activity 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');

WITH temp\_activity AS (

SELECT

username,

activity,

start\_date,

end\_date,

ROW\_NUMBER() OVER(PARTITION BY username ORDER BY start\_date, end\_date) row\_num,

COUNT(\*) OVER(PARTITION BY username ORDER BY start\_date, end\_date

rows between unbounded preceding and unbounded following) total\_activities

FROM

user\_activity

),

temp\_activity2 AS (

SELECT

username,

activity,

start\_date,

end\_date,

IF(total\_activities = 1 and row\_num = 1, 2, row\_num) AS ranking

FROM

temp\_activity

)

SELECT

username,

activity,

start\_date,

end\_date

FROM

temp\_activity2

WHERE

ranking = 2;

-- Q.102 SAME AS 101

-- Q.103 Query the name of any student in students who scored higher than 75 Marks. Order your output

-- by the last three characters of each name. If two or more students both have names ending in the same

-- last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending id.

CREATE TABLE students(

id INT,

name VARCHAR(20),

marks INT

);

INSERT INTO students VALUES

(1,'ASHLEY',81),

(2,'SAMANTHA',75),

(3,'JULIA',76),

(4,'BELVET',84);

SELECT

name

FROM

students

WHERE

marks > 75

ORDER BY

RIGHT(name,3),

id;

-- Q.104 Write a Query that prints a list of employee names (i.e.: the name attribute)

-- for employees in employee HAVING a salary greater than $2000 per month who have

-- been employees for less than 10 months. Sort your result by ascending employee\_id.

CREATE TABLE employee(

employee\_id INT,

name VARCHAR(20),

month INT,

salary INT

);

INSERT INTO employee VALUES

(12228,'ROSE',15,1968),

(33645,'ANGELA',1,3443),

(45692,'FRANK',17,1608),

(56118,'PATRICK',7,1345),

(59725,'LISA',11,2330),

(74197,'KIMBERLY',16,4372),

(78454,'BONNIE',8,1771),

(83565,'MICHAEL',6,2017),

(98607,"TODD",5,3396),

(99989,'JOE',9,3573);

SELECT

name

FROM

employee

WHERE

salary > 2000

AND

months < 10

ORDER BY

employee\_id;

-- Q.105 Write a Query identifying the type of each record in the TRIANGLES table using its three side lengths.

CREATE TABLE triangles(

a INT,

b INT,

c INT

);

INSERT INTO triangles VALUES

(20,20,23),

(20,20,20),

(20,21,22),

(13,14,30);

SELECT

a,

b,

c,

CASE

WHEN a + b <= c OR b + c <= a OR a + c <= b THEN 'NOT A TRIANGLE'

WHEN a = b AND b = c THEN 'EQUILATERAL'

WHEN a = b OR b = c OR c = a THEN 'ISSOCELES'

WHEN a <> b AND b <> c THEN 'SCALEAN'

END AS type\_of\_triangle

FROM

triangles;

-- Q.106 Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries),

-- and round it up to the next integer.

CREATE TABLE employees(

id INT,

name VARCHAR(20),

salary INT

);

INSERT INTO employees VALUES

(1,'KRISTEEN',1420),

(2,'ASHLEY',2006),

(3,'JULIA',2210),

(4,'MARIA',3000);

SELECT

ceil(avg(salary) - avg(replace(salary, '0', ''))) AS error

FROM

employees;

-- Q.107 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(20),

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

MAX(salary\*months) as total\_earnings,

COUNT(\*)

FROM

employee

WHERE

(salary\*months) in (

SELECT

MAX(months \* salary)

FROM

employee

);

-- Q.108 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. WHERE [occupation\_COUNT] is the number of occurrences of an occupation in OCCUPATIONS and [occupation]

-- is the lowerCASE occupation name. If more than one Occupation hAS the same [occupation\_COUNT],

-- they should be ordered alphabetically.

CREATE TABLE occupations(

name VARCHAR(20),

occupation VARCHAR(20)

);

INSERT INTO occupations VALUES

('SAMNATHA','DOCTOR'),

('JULIA','ACTOR'),

('MARIA','ACTOR'),

('MEERA','SINGER'),

('ASHLEY','PROFESSOR'),

('KETTY','PROFESSOR'),

('CHRISTEEN','PROFESSOR'),

('JANE','ACTOR'),

('JENNY','DOCTOR'),

('PRIYA','SINGER');

-- a.

SELECT

CONCAT(name, '(',substring(occupation, 1, 1),')') as `name(occupation)`

FROM

occupations

ORDER BY

name;

-- b.

SELECT

CONCAT("There are a total of ",

COUNT(\*),' ', lower(occupation), 's.') AS info

FROM

occupations

GROUP BY

occupation

ORDER BY

COUNT(occupation),

occupation;

-- Q.109 Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically

-- and displayed underneath its corresponding Occupation. The output column headers should be

-- Doctor, Professor, Singer, and Actor, respectively.

CREATE TABLE occupations(

name VARCHAR(20),

occupation VARCHAR(20)

);

INSERT INTO occupations VALUES

('SAMNATHA','DOCTOR'),

('JULIA','ACTOR'),

('MARIA','ACTOR'),

('MEERA','SINGER'),

('ASHLEY','PROFESSOR'),

('KETTY','PROFESSOR'),

('CHRISTEEN','PROFESSOR'),

('JANE','ACTOR'),

('JENNY','DOCTOR'),

('PRIYA','SINGER');

SELECT

MAX(CASE WHEN occupation = 'Doctor' then name END) AS Doctor,

MAX(CASE WHEN occupation = 'Professor' then name END) AS Professor,

MAX(CASE WHEN occupation = 'Singer' then name END) AS Singer,

MAX(CASE WHEN occupation = 'Actor' then name END) AS Actor

FROM

(

SELECT

name,

occupation,

row\_number() over(partition by occupation order by name) AS row\_num

FROM

occupations

) AS base

GROUP BY

row\_num;

-- Q.110 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 n NOT IN (SELECT DISTINCT p FROM bst WHERE p IS NOT NULL) THEN 'Leaf'

WHEN p IS NULL THEN 'Root'

ELSE 'Inner'

END AS type

FROM

bst

ORDER BY

n;

-- Q.111 Given the table schemas below, write a query to print the company\_code,

-- founder name, total number of lead managers, total number of senior managers,

-- total number of managers, and total number of employees. Order your output by

-- ascending company\_code.

CREATE TABLE company(

company\_code VARCHAR(20),

founder VARCHAR(20)

);

CREATE TABLE lead\_manager(

lead\_manager\_code VARCHAR(20),

company\_code VARCHAR(20)

);

CREATE TABLE senior\_manager(

senior\_manager\_code VARCHAR(20),

lead\_manager\_code VARCHAR(20),

company\_code VARCHAR(20)

);

CREATE TABLE manager(

manager\_code VARCHAR(20),

senior\_manager\_code VARCHAR(20),

lead\_manager\_code VARCHAR(20),

company\_code VARCHAR(20)

);

CREATE TABLE employee(

employee\_code VARCHAR(20),

manager\_code VARCHAR(20),

senior\_manager\_code VARCHAR(20),

lead\_manager\_code VARCHAR(20),

company\_code VARCHAR(20)

);

INSERT INTO company VALUES

('C1','MONIKA'),

('C2','SAMANTHA');

INSERT INTO lead\_manager VALUES

('LM1','C1'),

('LM2','C2');

INSERT INTO senior\_manager VALUES

('SM1','LM1','C1'),

('SM2','LM1','C1'),

('SM3','LM2','C2');

INSERT INTO manager VALUES

('M1','SM1','LM1','C1'),

('M2','SM3','LM2','C2'),

('M3','SM3','LM2','C2');

INSERT INTO employee VALUES

('E1','M1','SM1','LM1','C1'),

('E2','M1','SM1','LM1','C1'),

('E3','M2','SM3','LM2','C2'),

('E4','M3','SM3','LM2','C2');

SELECT

c.company\_code,

c.founder,

COUNT(DISTINCT lm.lead\_manager\_code),

COUNT(DISTINCT sm.senior\_manager\_code),

COUNT(DISTINCT m.manager\_code),

COUNT(DISTINCT e.employee\_code)

FROM

company c

INNER JOIN

lead\_manager lm

ON

c.company\_code = lm.company\_code

INNER JOIN

senior\_manager sm

ON

sm.lead\_manager\_code = lm.lead\_manager\_code

INNER JOIN

manager m

ON

m.senior\_manager\_code = sm.senior\_manager\_code

INNER JOIN

employee e

ON

e.manager\_code = m.manager\_code

GROUP BY

c.company\_code, c.founder

ORDER BY

c.company\_code;

-- Q.112 Write a query to print all prime numbers less than or equal to 1000.

-- Print your result on a single line, and use the ampersand () character as

-- your separator (instead of a space).

WITH RECURSIVE number\_generation AS (

SELECT

1 num

UNION ALL

SELECT

num + 1

FROM

number\_generation

WHERE

num<1000

),

number\_generation2 AS (

SELECT

n1.num AS numm

FROM

number\_generation n1

INNER JOIN

number\_generation n2

WHERE

n1.num % n2.num = 0

GROUP BY

n1.num

HAVING

COUNT(n1.num) = 2

)

SELECT

group\_concat(numm ORDER BY numm SEPARATOR '&') AS prime\_numbers

FROM

number\_generation2;

-- Q.113 Write a query to print the pattern P(20).

WITH RECURSIVE generate\_numbers AS

(

SELECT

1 AS n

UNION

SELECT

n+1

FROM

generate\_numbers

WHERE

n<20

)

SELECT

repeat('\*',n)

FROM

generate\_numbers;

-- Q.114 Write a query to print the pattern P(20).

WITH RECURSIVE generate\_numbers AS

(

SELECT

20 AS n

UNION

SELECT

n-1

FROM

generate\_numbers

WHERE

n>1

)

SELECT

repeat('\*',n)

FROM

generate\_numbers;

-- Q.115 SAME AS Q.103

-- Q.116 SAME AS Q.79

-- Q.117 SAME AS Q.104

-- Q.118 SAME AS Q.105

-- Q.119 SAME AS Q.80

-- Q.120 SAME AS Q.81

-- Q.121 SAME AS Q.82

-- Q.122 SAME AS Q.83

-- Q.123 SAME AS Q.84

-- Q.124 SAME AS Q.85

-- Q.125 SAME AS Q.86

-- Q.126 SAME AS Q.87

-- Q.127 SAME AS Q.68

-- Q.128 SAME AS Q.55

-- Q.129 SAME AS Q.90

-- Q.130 SAME AS Q.91

-- Q.131 SAME AS Q.92

-- Q.132 SAME AS Q.50

-- Q.133 SAME AS Q.94

-- Q.134 SAME AS Q.94

-- Q.135 SAME AS Q.101

-- Q.136 SAME AS Q.101

-- Q.137 SAME AS Q.106

-- Q.138 SAME AS Q.105

-- Q.139 SAME AS Q.105

-- Q.140 SAME AS Q.105

-- Q.141 SAME AS Q.110

-- Q.142 SAME AS Q.111

-- Q.143 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 functions(

x INT,

y INT

);

INSERT INTO functions VALUES

(20,20),

(20,20),

(20,21),

(23,22),

(22,23),

(21,20);

WITH temp\_functions AS (

SELECT

x,

y,

ROW\_NUMBER() OVER (ORDER BY x, y) AS row\_num

FROM

functions

)

SELECT

DISTINCT f1.x,

f1.y

FROM

temp\_functions f1

INNER JOIN

temp\_functions f2

ON

f1.x = f2.y

AND

f1.y = f2.x

AND

f1.row\_num <> f2.row\_num

WHERE

f1.x <= f1.y

ORDER BY

f1.x;

-- Q.144 Write a query to output the names of those students whose best friENDs got offered a higher

-- salary than them. Names must be ordered by the salary amount offered to the best friENDs.

-- It is guaranteed that no two students get the same salary offer.

CREATE TABLE students(

id INT,

name VARCHAR(20)

);

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

(1,15.20),

(2,10.06),

(3,11.55),

(4,12.12);

SELECT

s1.name

FROM

friends f1

INNER JOIN

students s1

ON

f1.id = s1.id

INNER JOIN

students s2

ON

f1.friend\_id = s2.id

INNER JOIN

packages p1

ON

f1.id = p1.id

INNER JOIN

packages p2

ON

f1.friend\_id = p2.id

WHERE

p1.salary < p2.salary

ORDER BY

p2.salary;

-- Q.145 Write a query to print the respective hacker\_id and name of hackers who achieved full scores for

-- more than one challenge. Order your output in descENDing order by the total number of challenges in

-- which the hacker earned a full score. If more than one hacker received full scores in the same number

-- of challenges, then sort them by ascending hacker\_id.

CREATE TABLE hackers(

hacker\_id INT,

name VARCHAR(20)

);

CREATE TABLE difficulty(

difficulty\_level INT,

score INT

);

CREATE TABLE challenges(

challenge\_id INT,

hacker\_id INT,

difficulty\_level INT

);

CREATE TABLE submissions(

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),

(26566,5580,7),

(66730,52243,6),

(71055,52243,2);

SELECT

h.hacker\_id,

h.name

FROM

hackers h

INNER JOIN

submissions s

ON

h.hacker\_id = s.hacker\_id

INNER JOIN

challenges c

ON

s.challenge\_id = c.challenge\_id

INNER JOIN

difficulty d

ON

c.difficulty\_level = d.difficulty\_level

WHERE

s.score = d.score

GROUP BY

h.name, h.hacker\_id

HAVING

COUNT(s.score) > 1

ORDER BY

COUNT(s.challenge\_id) desc,

h.hacker\_id;

-- Q.146 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');

-- Approach 1

WITH project\_start AS

(

SELECT

start\_date,

ROW\_NUMBER() OVER() AS ps\_rownum

FROM

projects

WHERE start\_date not in (

SELECT

end\_date

FROM

projects

)

),

project\_end AS

(

SELECT

end\_date,

ROW\_NUMBER() OVER() AS pe\_rownum

FROM

projects

WHERE

END\_date not in (

SELECT

start\_date

FROM

projects

)

)

SELECT

project\_start.start\_date,

project\_end.end\_date

FROM

project\_start

INNER JOIN

project\_end

on

project\_end.pe\_rownum = project\_start.ps\_rownum

ORDER BY

DATEDIFF(project\_start.start\_date, project\_end.end\_date) desc,

project\_start.start\_date;

-- Approach 2

WITH temp\_project AS (

SELECT

temp.start\_date,

temp.end\_date,

SUM(

CASE

WHEN previous\_end\_date IS NULL THEN 0

WHEN DAY(end\_date) - DAY(previous\_end\_date) = 1 THEN 0

ELSE 1

END

) OVER(ORDER BY start\_date) AS project\_num

FROM (

SELECT

start\_date,

end\_date,

LAG(end\_date) OVER (ORDER BY start\_date) AS previous\_end\_date

FROM

projects

) temp

)

SELECT

MIN(start\_date) AS project\_start\_date,

MAX(end\_date) as project\_end\_date

FROM

temp\_project

GROUP BY

project\_num

ORDER BY

DAY(MAX(end\_date))-DAY(MIN(start\_date));

-- Approach 3

WITH temp\_project AS (

SELECT

temp.start\_date,

temp.end\_date,

SUM(

CASE

WHEN previous\_end\_date IS NULL THEN 0

WHEN DAY(end\_date) - DAY(previous\_end\_date) = 1 THEN 0

ELSE 1

END

) over(order by start\_date range between unbounded preceding and current row) AS project\_num

FROM (

SELECT

start\_date,

end\_date,

LAG(end\_date) OVER (ORDER BY start\_date) AS previous\_end\_date

FROM

projects

) temp

)

SELECT

MIN(start\_date) AS project\_start\_date,

MAX(end\_date) AS project\_end\_date

FROM

temp\_project

GROUP BY

project\_num

ORDER BY

DAY(MAX(end\_date))-DAY(MIN(start\_date));

-- Q.147 In an effort to identify high-value customers, Amazon asked for your help to obtain data

-- about users who go on shopping sprees. A shopping spree occurs when a user makes purchASes on 3

-- or more consecutive days. List the user IDs who have gone on at leASt 1 shopping spree in ascending order.

CREATE TABLE transactions(

user\_id INT,

amount FLOAT,

transaction\_date DATETIME

);

INSERT INTO transactions VALUES

(1,9.99,'08/01/2022 10:00:00'),

(1,55,'08/17/2022 10:00:00'),

(2,149.5,'08/05/2022 10:00:00'),

(2,4.89,'08/06/2022 10:00:00'),

(2,34,'08/07/2022 10:00:00');

SELECT

DISTINCT user\_id

FROM

(

SELECT

user\_id,

transaction\_date,

rn,

transaction\_date :: date - rn::integer,

COUNT(transaction\_date :: date - rn::integer) OVER(PARTITION BY user\_id) AS cn

FROM

(

SELECT

user\_id,

transaction\_date,

ROW\_NUMBER() OVER(PARTITION BY user\_id ORDER BY transaction\_date) AS rn

FROM

transactions

) temp

)temp1

WHERE cn >=3;

-- SOLVED IN POSTGRESQL

-- Q.148 You are given a table of PayPal payments showing the payer, the recipient, and the amount paid.

-- A two-way unique relationship is established WHEN two people sEND money back and forth. Write a query

-- to find the number of two-way unique relationships in this data.

CREATE TABLE payments(

payer\_id INT,

recipient\_id INT,

amount INT

);

INSERT INTO payments VALUES

(101,201,30),

(201,101,10),

(101,301,20),

(301,101,80),

(201,301,70);

WITH temp\_payments AS (

SELECT

DISTINCT p1.payer\_id,

p1.recipient\_id

FROM

payments p1

INNER JOIN

payments p2

ON

p1.payer\_id = p2.recipient\_id

AND

p2.payer\_id = p1.recipient\_id

AND

p1.payer\_id < p2.payer\_id

)

SELECT

COUNT(\*) unique\_relationships

FROM

temp\_payments;

-- Q.149 Write a query to obtain the list of customers whose first transaction was valued at $50 or more.

-- Output the number of users.

CREATE TABLE user\_transactions(

transaction\_id INT,

user\_id INT,

spend FLOAT,

transaction\_date VARCHAR(30)

);

INSERT INTO user\_transactions VALUES

(759274,111,49.50,'02/03/2022 00:00:00'),

(850371,111,51.00,'03/15/2022 00:00:00'),

(615348,145,36.30,'03/22/2022 00:00:00'),

(137424,156,151.00,'04/04/2022 00:00:00'),

(248475,156,87.00,'04/16/2022 00:00:00');

WITH temp\_transactions AS (

SELECT

user\_id,

transaction\_date,

spend,

ROW\_NUMBER() OVER(PARTITION BY user\_id ORDER BY transaction\_date) row\_num

FROM

user\_transactions

)

SELECT

COUNT(DISTINCT user\_id) as users

FROM

temp\_transactions

WHERE

row\_num = 1

and

spend >= 50;

-- Q.150 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 measurments(

measurment\_id INT,

measurment\_value FLOAT,

measurment\_time DATETIME

);

INSERT INTO measurments VALUES

(131233,1109.51,'07/10/2022 09:00:00'),

(135211,1662.74,'07/10/2022 11:00:00'),

(523542,1246.24,'07/10/2022 13:15:00'),

(143562,1124.50,'07/11/2022 15:00:00'),

(346462,1234.14,'07/11/2022 16:45:00');

WITH temp\_measurments AS (

SELECT

measurement\_value,

measurement\_time,

ROW\_NUMBER() OVER(PARTITION BY measurement\_time::DATE ORDER BY measurement\_time) row\_num

FROM

measurments

)

SELECT

measurement\_time::DATE,,

ROUND(SUM(

CASE

WHEN row\_num % 2 <> 0 THEN measurment\_value

END),2) AS odd\_value,

ROUND(SUM(

CASE

WHEN row\_num % 2 = 0 THEN measurment\_value

END),2) AS even\_value

FROM

temp\_measurments

GROUP BY

measurement\_time::DATE,

ORDER BY

measurement\_time;

-- SOLVED IN POSTGRESQL

-- Q.151 SAME AS Q.147

-- Q.152 The Airbnb Booking RecommENDations team is trying to understand the "substitutability" of two rentals

-- and whether one rental is a good substitute for another. They want you to write a query to find the unique

-- combination of two Airbnb rentals WITH the same exact amenities offered. Output the COUNT of the unique

-- combination of Airbnb rentals.

CREATE TABLE rental\_amenities(

rental\_id INT,

amenity VARCHAR(20)

);

INSERT INTO rental\_amenities VALUES

(123,'POOL'),

(123,'KITCHEN'),

(234,'HOT TUB'),

(234,'FIREPLACE'),

(345,'KITCHEN'),

(345,'POOL'),

(456,'POOL');

WITH temp\_amenities AS (

SELECT

rental\_id,

amenity,

COUNT(amenity) over(partition by rental\_id) AS no\_of\_amenities

FROM

rental\_amenities

),

temp\_amenities2 AS (

SELECT

COUNT(\*)

FROM

temp\_amenities a

inner join

temp\_amenities b

on

a.no\_of\_amenities = b.no\_of\_amenities

AND

a.amenity = b.amenity

AND

a.rental\_id<>b.rental\_id

GROUP BY

a.rental\_id,

b.rental\_id,

a.no\_of\_amenities

HAVING

COUNT(\*) = a.no\_of\_amenities

)

SELECT

CEIL(COUNT(\*)/2) as matching\_airbnb

FROM

temp\_amenities2;

-- Q.153 Write a query to calculate the return on ad spend (ROAS) for each advertiser

-- across all ad campaigns. Round your answer to 2 decimal places, and order your

-- output by the advertiser\_id.

CREATE TABLE ad\_campaigns(

campaign\_id INT,

spend INT,

revenue FLOAT,

advertiser\_id INT

);

INSERT INTO ad\_campaigns VALUES

(1,5000,7500,3),

(2,1000,900,1),

(3,3000,12000,2),

(4,500,200,4),

(5,100,400,4);

SELECT

advertiser\_id,

CAST(SUM(revenue) / SUM(spend) AS Decimal(8,2)) AS ROAS

FROM

ad\_campaigns

GROUP BY

advertiser\_id

ORDER BY

advertiser\_id;

-- Q.154 Write a query that shows the following data for each compensation outlier:

-- employee ID, salary, and whether they are potentially overpaid or potentially underpaid

CREATE TABLE employee\_pay(

employee\_id INT,

salary INT,

title VARCHAR(20)

);

INSERT INTO employee\_pay VALUES

(101,80000,'DATA ANALYST'),

(102,90000,'DATA ANALYST'),

(103,100000,'DATA ANALYST'),

(104,30000,'DATA ANALYST'),

(105,120000,'DATA SCIENTIST'),

(106,100000,'DATA SCIENTIST'),

(107,80000,'DATA SCIENTIST'),

(108,310000,'DATA SCIENTIST');

WITH temp\_compensation AS (

SELECT

employee\_id,

salary,

title,

ROUND(AVG(salary) over(PARTITION BY title),2) AS avg\_salary

FROM

employee\_pay

),

temp\_compensation2 AS (

SELECT

employee\_id,

salary,

CASE

WHEN salary > 2 \* avg\_salary THEN 'Overpaid'

WHEN salary < avg\_salary/2 THEN 'Underpaid'

END AS status

FROM

temp\_compensation

)

SELECT

employee\_id,

salary,

status

FROM

temp\_compensation2

WHERE

status is not null;

-- Q.155 SAME AS 148

-- Q.156 Assume you are given the table below containing information on user

-- purchASes. Write a query to obtain the number of users who purchASed the

-- same product on two or more different days. Output the number of unique users.

CREATE TABLE purchases(

user\_id INT,

product\_id INT,

quantity INT,

purchase\_date DATETIME

);

INSERT INTO purchases VALUES

(536,3223,6,'2022-01-11 12:33:44'),

(827,3585,35,'2022-02-20 14:05:26'),

(536,3223,5,'2022-03-02 09:33:28'),

(536,1435,10,'2022-03-02 08:40:00'),

(827,2452,45,'2022-04-09 00:00:00');

SELECT

COUNT(DISTINCT user\_id) AS repeat\_purchasers

FROM (

SELECT

user\_id

FROM

purchases

GROUP BY

user\_id,

product\_id

HAVING

COUNT(DISTINCT purchase\_date) > 1

) temp;

-- Q.157 Say you have access to all the transactions for a given merchant acCOUNT.

-- Write a query to print the cumulative balance of the merchant acCOUNT at the END

-- of each day, WITH the total balance reset back to zero at the END of the month.

-- Output the transaction date and cumulative balance.

CREATE TABLE transactions(

transaction\_id INT,

type ENUM('DEPOSIT','WITHDRAWL'),

amount FLOAT,

transaction\_date DATETIME

);

INSERT INTO transactions VALUES

(19153,'DEPOSIT',65.90,'07/10/2022 10:00:00'),

(53151,'DEPOSIT',178.55,'07/08/2022 10:00:00'),

(29776,'WITHDRAWL',25.90,'07/08/2022 10:00:00'),

(16461,'WITHDRAWL',45.99,'07/08/2022 10:00:00'),

(77134,'DEPOSIT',32.60,'07/10/2022 10:00:00');

SELECT

DISTINCT date(transaction\_date),

SUM(

CASE

WHEN type = 'deposit' THEN amount

ELSE -amount

END

) OVER(PARTITION BY EXTRACT(MONTH FROM transaction\_date) ORDER BY DATE(transaction\_date)) AS balance

FROM

transactions;

-- SOLVED IN POSTGRESQL

-- Q.158 Assume you are given the table below containing information on

-- Amazon customers and their spend on products belonging to various

-- categories. Identify the top two highest-grossing products within each

-- category in 2022. Output the category, product, and total spend.

CREATE TABLE product\_spend(

category VARCHAR(20),

product VARCHAR(20),

user\_id INT,

spend FLOAT,

transaction\_date DATETIME

);

INSERT INTO product\_spend VALUES

('APPLIANCE','REFRIGERATOR',165,246.00,'12/26/2021 12:00:00'),

('APPLIANCE','REFRIGERATOR',123,299.99,'03/02/2022 12:00:00'),

('APPLIANCE','WASHING MACHINE',123,219.80,'03/02/2022 12:00:00'),

('ELECTRONICS','VACUUM',178,152.00,'04/05/2022 12:00:00'),

('ELECTRONICS','WIRELESS HEADSET',156,249.90,'07/08/2022 12:00:00'),

('ELECTRONICS','REFRIGERATOR',145,189.00,'07/15/2022 12:00:00');

WITH temp\_product\_details AS (

SELECT

category,

product,

spend,

SUM(spend) OVER(PARTITION BY category, product) total\_spend

FROM

product\_spend

WHERE

EXTRACT(YEAR FROM transaction\_date) = 2022

),

temp\_product\_details1 AS (

SELECT

DISTINCT category,

product,

total\_spend ,

DENSE\_RANK() OVER(PARTITION BY category order by total\_spend desc) row\_num

FROM

temp\_product\_details

)

SELECT

DISTINCT category,

product,

total\_spend

FROM

temp\_product\_details1

WHERE

row\_num <=2

ORDER BY

category,

total\_spend DESC;

-- SOLVED IN POSTGRESQL

-- Q.159 Facebook is analysing its user signup data for June 2022.

-- Write a query to generate the churn rate by week in June 2022.

-- Output the week number (1, 2, 3, 4, ...) and the corresponding

-- churn rate rounded to 2 decimal places.

CREATE TABLE users(

user\_id INT,

signup\_date DATETIME,

last\_login DATETIME

);

INSERT INTO users VALUES

(1001,'2022-06-01 12:00:00','2022-07-05 12:00:00'),

(1002,'2022-06-03 12:00:00','2022-06-15 12:00:00'),

(1004,'2022-06-02 12:00:00','2022-06-15 12:00:00'),

(1006,'2022-06-15 12:00:00','2022-06-27 12:00:00'),

(1012,'2022-06-16 12:00:00','2022-07-22 12:00:00');

-- Approach 1

WITH temp\_churn\_rate AS (

SELECT

user\_id,

signup\_date,

last\_login,

DATEDIFF(last\_login, signup\_date) diff,

EXTRACT(WEEK FROM signup\_date) AS week\_no,

WEEK(signup\_date,5) - WEEK(DATE\_SUB(signup\_date, INTERVAL DAYOFMONTH(signup\_date)-1 DAY),5)+1 AS ranking

FROM

users

WHERE

EXTRACT(MONTH FROM signup\_date) = 6

AND

EXTRACT(YEAR FROM signup\_date) = 2022

),

temp\_churn\_rate2 AS (

SELECT

ranking,

COUNT(ranking) AS total\_users,

COUNT(

CASE

WHEN diff <= 28 THEN 1

END

) AS total\_churns

FROM

temp\_churn\_rate

GROUP BY

ranking

)

SELECT

ranking AS week,

ROUND((total\_churns/total\_users) \* 100 ,2) AS churn\_rate

FROM

temp\_churn\_rate2

ORDER BY

ranking;

-- Approach 2

WITH temp\_churn\_rate AS (

SELECT

user\_id,

signup\_date,

last\_login,

DATEDIFF(last\_login, signup\_date) diff,

EXTRACT(WEEK FROM signup\_date) AS week\_no,

DENSE\_RANK() OVER(ORDER BY EXTRACT(WEEK FROM signup\_date)) ranking

FROM

users

WHERE

EXTRACT(MONTH FROM signup\_date) = 6

AND

EXTRACT(YEAR FROM signup\_date) = 2022

),

temp\_churn\_rate2 AS (

SELECT

ranking,

COUNT(ranking) AS total\_users,

COUNT(

CASE

WHEN diff <= 28 THEN 1

END

) AS total\_churns

FROM

temp\_churn\_rate

GROUP BY

ranking

)

SELECT

ranking AS week,

ROUND((total\_churns/total\_users) \* 100 ,2) AS churn\_rate

FROM

temp\_churn\_rate2

ORDER BY

ranking;