```
--Solution_of_SQL_Class_1_Assignment

create database gds_mysql_assignment_db1;

use gds_mysql_assignment_db1;


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

create table CITY
(
    ID int,
    NAME VARCHAR(17),
    COUNTRYCODE VARCHAR(3),
    DISTRICT VARCHAR(20),
    POPULATION int
);

describe CITY;


insert into CITY VALUES(6,'Rotterdam','NLD','Zuid-Holland',593321);
insert into CITY VALUES(3878,'Scottsdale','USA','Arizona',202705);
insert into CITY VALUES(3965,'Corona','USA','California',124966);
insert into CITY VALUES(3973,'Concord','USA','California',121780);
insert into CITY VALUES(3977,'Cedar Rapids','USA','Iowa',120758);
insert into CITY VALUES(3982,'Coral Springs','USA','Florida',117549);
insert into CITY VALUES(4054,'Fairfield','USA','California',92256);
insert into CITY VALUES(4058,'Boulder','USA','Colorado',91238);
insert into CITY VALUES(4061,'Fall River','USA','Massachusetts',90555);

select * from CITY;

SELECT ID,NAME,COUNTRYCODE,DISTRICT,POPULATION FROM CITY WHERE
COUNTRYCODE = 'USA' AND POPULATION > 100000;
SELECT * FROM CITY WHERE COUNTRYCODE = 'USA' AND POPULATION > 100000;



--Q2. Query the NAME field for all American cities in the CITY table
with populations larger than 120000.
--The CountryCode for America is USA.
--The CITY table is described as follows:
SELECT NAME FROM CITY WHERE COUNTRYCODE = 'USA' AND POPULATION >
120000;


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

```
--The CITY table is described as follows:
select * from CITY;


--Q4. Query all columns for a city in CITY with the ID 1661.
--The CITY table is described as follows:
select * from CITY where ID = 1661;


--Q5. Query all attributes of every Japanese city in the CITY table.
The COUNTRYCODE for Japan is JPN.
--The CITY table is described as follows:

SELECT * FROM CITY WHERE COUNTRYCODE = 'JPN';


--Q6. Query the names of all the Japanese cities in the CITY table. The
COUNTRYCODE for Japan is JPN.
--The CITY table is described as follows:

SELECT NAME FROM CITY WHERE COUNTRYCODE = 'JPN';



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

create table if not exists STATION
(
     ID INT,
     CITY VARCHAR(21),
     STATE VARCHAR(2),
     LAT_N INT,
     LONG_W INT
);

DESCRIBE STATION;


INSERT INTO STATION VALUES(794,'Kissee Mills','MO',139,73);
INSERT INTO STATION VALUES(824,'Loma Mar','CA',48,130);
INSERT INTO STATION VALUES(603,'Sandy Hook','CT',72,148);
INSERT INTO STATION VALUES(478,'Tipton','IN',33,97);
INSERT INTO STATION VALUES(619,'Arlington','CO',75,92);
INSERT INTO STATION VALUES(711,'Turner','AR',50,101);
INSERT INTO STATION VALUES(839,'Slidell','LA',85,151);
INSERT INTO STATION VALUES(411,'Negreet','LA',98,105);
INSERT INTO STATION VALUES(588,'Glencoe','KY',46,136);
INSERT INTO STATION VALUES(665,'Chelsea','IA',98,59);
INSERT INTO STATION VALUES(342,'Chignik Lagoon','AK',103,153);
INSERT INTO STATION VALUES(733,'Pelahatchie','MS',38,28);
INSERT INTO STATION VALUES(811,'Dorrance','KS',102,121);
```

```sql
INSERT INTO STATION VALUES(698,'Albany','CA',49,80);
INSERT INTO STATION VALUES(325,'Monument','KS',70,141);
INSERT INTO STATION VALUES(414,'Manchester','MD',73,37);
INSERT INTO STATION VALUES(113,'Prescott','IA',39,65);
INSERT INTO STATION VALUES(971,'Graettinger','IA',94,150);
INSERT INTO STATION VALUES(266,'Cahone','CO',116,127);


SELECT * FROM STATION;


SELECT CITY,STATE FROM STATION;


--Q8. Query a list of CITY names from STATION for cities that have an
even ID number. Print the results in any order, but exclude duplicates
from the answer.
SELECT DISTINCT(CITY) AS City_Name FROM STATION WHERE ID%2 = 0 ORDER BY
CITY ASC;


--Q9. Find the difference between the total number of CITY entries in
the table and the number of distinct CITY entries in the table.
--For example, if there are three records in the table with CITY values
'New York', 'New York', 'Bengalaru',
--there are 2 different city names: 'New York' and 'Bengalaru'. The
query returns , because total number
--of records - number of unique city names = 3-2 =1
SELECT COUNT(CITY) AS TOTAL_NUMBER_OF_RECORDS,COUNT(DISTINCT(CITY)) AS
NUMBER_OF_UNIQUE_CITY_NAMES,(COUNT(CITY) - COUNT(DISTINCT(CITY))) AS
DIFFERENCE_CITY_COUNT  FROM STATION;


--Q10. Query the two cities in STATION with the shortest and longest
CITY names, as well as their respective lengths (i.e.: number of
characters in the name). If there is more than one smallest or largest
city, choose the one that comes first when ordered alphabetically.

--Sample Input
--For example, CITY has four entries: DEF, ABC, PQRS and WXY.
--Sample Output
--ABC 3
--PQRS 4
--Hint -
--When ordered alphabetically, the CITY names are listed as ABC, DEF,
PQRS, and WXY, with lengths and. The longest name is PQRS, but there
are options for shortest named city. Choose ABC, because it comes first
alphabetically.
--Note
--You can write two separate queries to get the desired output. It need
not be a single query.
```

```
SELECT CITY,LENGTH(CITY)AS MIN_LENGTH_OF_CITY FROM STATION ORDER BY
LENGTH(CITY) ASC LIMIT 1;

SELECT CITY,LENGTH(CITY) AS MAX_LENGTH_OF_CITY FROM STATION ORDER BY
LENGTH(CITY) DESC LIMIT 1;


--Q11. Query the list of CITY names starting with vowels (i.e., a, e,
i, o, or u) from STATION. Your result cannot contain duplicates.
SELECT DISTINCT(CITY) AS DISTINCT_CITY_NAME FROM STATION WHERE
lower(SUBSTR(city,1,1)) in ('a','e','i','o','u');

SELECT DISTINCT CITY FROM STATION
WHERE lcase(CITY) LIKE 'a%'
OR lcase(CITY) LIKE 'e%'
OR lcase(CITY) LIKE 'i%'
OR lcase(CITY) LIKE 'o%'
OR lcase(CITY) LIKE 'u%'
ORDER BY CITY;

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

SELECT DISTINCT CITY FROM STATION
WHERE lcase(CITY) LIKE '%a'
OR lcase(CITY) LIKE '%e'
OR lcase(CITY) LIKE '%i'
OR lcase(CITY) LIKE '%o'
OR lcase(CITY) LIKE '%u'
ORDER BY CITY;



--Q13. Query the list of CITY names from STATION that do not start with
vowels. Your result cannot contain duplicates.
SELECT DISTINCT CITY FROM STATION
WHERE lcase(CITY) NOT LIKE 'a%'
AND lcase(CITY) NOT LIKE 'e%'
AND lcase(CITY) NOT LIKE 'i%'
AND lcase(CITY) NOT LIKE 'o%'
AND lcase(CITY) NOT LIKE 'u%'
ORDER BY CITY;


--Q14.Query the list of CITY names from STATION that do not end with
vowels. Your result cannot contain duplicates.
SELECT DISTINCT CITY FROM STATION
WHERE lcase(CITY) NOT LIKE '%a'
AND lcase(CITY) NOT LIKE '%e'
AND lcase(CITY) NOT LIKE '%i'
AND lcase(CITY) NOT LIKE '%o'
AND lcase(CITY) NOT LIKE '%u'
ORDER BY CITY;


--Q15. Query the list of CITY names from STATION that either do not
start with vowels or do not end with vowels. Your result cannot contain
duplicates.

select distinct CITY from STATION where CITY not regexp '^[aeiou]' or
city not regexp '[aeiou]$';


--Q16. Query the list of CITY names from STATION that do not start with
vowels and do not end with vowels. Your result cannot contain
duplicates.
select distinct CITY from STATION where CITY not regexp '^[aeiou]' AND
city not regexp '[aeiou]$';


--Q17.
--Table: Product
--product_id is the primary key of this table.
--Each row of this table indicates the name and the price of each
product.
create table if not exists Product
(
    product_id int NOT NULL,
    product_name varchar(50),
    unit_price int,
    constraint pk PRIMARY KEY (product_id)
);


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

select * from Product;

--Table: Sales
create table if not exists Sales
(
    seller_id int,
    product_id int,
    buyer_id int,
    sale_date date,
    quantity int,
    price int,
    constraint fk FOREIGN KEY (product_id) REFERENCES
Product(product_id)
);
```

```
insert into Sales values(1,1,1,'2019-01-21',2,2000);
insert into Sales values(1,2,2,'2019-02-17',1,800);
insert into Sales values(2,2,3,'2019-06-02',1,800);
insert into Sales values(3,3,4,'2019-05-13',2,2800);

select * from Sales;


--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.
--Explanation:
--The product with id 1 was only sold in the spring of 2019.
--The product with id 2 was sold in the spring of 2019 but was also
sold after the spring of 2019.
--The product with id 3 was sold after spring 2019.
--We return only product 1 as it is the product that was only sold in
the spring of 2019.

SELECT product_id,
       product_name
FROM   Product
WHERE  product_id NOT IN (SELECT product_id
                          FROM   Sales
                          WHERE  sale_date NOT BETWEEN
                                 '2019-01-01' AND '2019-03-31');



--Q18.
--Table: Views

create table if not exists Views
(
    article_id int,
    author_id int,
    viewer_id int,
    view_date date
);


insert into Views VALUES (1,3,5,'2019-08-01'),
(1,3,6,'2019-08-02'),(2,7,7,'2019-08-01'),(2,7,6,'2019-08-02'),
(4,7,1,'2019-07-22'), (3,4,4,'2019-07-21'),(3,4,4,'2019-07-21');

select * from Views;


--There is no primary key for this table, it may have duplicate rows.
```

```
--Each row of this table indicates that some viewer viewed an article
(written by some author) on some date.
--Note that equal author_id and viewer_id indicate the same person.
--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.
--The query result format is in the following example.

select distinct author_id as id from Views where author_id = viewer_id
order by author_id asc;



--Q19.
--Table: Delivery
create table if not exists Delivery
(
    delivery_id int not null,
    customer_id int,
    order_date date,
    customer_pref_delivery_date date,
    constraint pk PRIMARY KEY (delivery_id)
);


insert into Delivery values
(1,1,'2019-08-01','2019-08-02'),(2,5,'2019-08-02','2019-08-02'),(3,1,'2
019-08-11','2019-08-11'),(4,3,'2019-08-24','2019-08-26'),(5,4,'2019-08-
21','2019-08-22'),(6,2,'2019-08-11','2019-08-13');

select * from Delivery;

--The table holds information about food delivery to customers that
make orders at some date and
--specify a preferred delivery date (on the same order date or after
it).
--If the customer's preferred delivery date is the same as the order
date, then the order is called  immediately; otherwise, it is called
scheduled.
--Write an SQL query to find the percentage of immediate orders in the
table, rounded to 2 decimal places.
--The query result format is in the following example.

select round(100*d2.immediate_orders/count(d1.delivery_id), 2) as
immediate_percentage
from Delivery d1,
    (select count(order_date) as immediate_orders
    from Delivery
    where (order_date = customer_pref_delivery_date)) d2;
```

```
--Q20.
--Table: Ads
create table if not exists Ads
(
    ad_id int,
    user_id int,
    action enum('Clicked', 'Viewed', 'Ignored'),
    constraint pk PRIMARY KEY (ad_id, user_id)
);

insert into Ads VALUES
(1,1,'Clicked'),(2,2,'Clicked'),(3,3,'Viewed'),(5,5,'Ignored'),(1,7,'Ig
nored'),(2,7,'Viewed'),(3,5,'Clicked'),(1,4,'Viewed'),(2,11,'Viewed'),(
1,2,'Clicked');

select * from Ads;


--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.

select ad_id,
ifnull(
    round(
        avg(
            case
                when action = "Clicked" then 1
                when action = "Viewed" then 0
                else null
            end
        ) * 100,
    2),
0)
as ctr
from Ads
group by ad_id
order by ctr desc, ad_id asc;



--Q21.
--Table: Employee

create table if not exists Employee
(
    employee_id int,
    team_id int,
    constraint pk PRIMARY KEY (employee_id)
);
```

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

select * from Employee;

--Write an SQL query to find the team size of each of the employees.
--Return result table in any order.
select employee_id,
         count(*) over(partition by team_id) as team_size
from Employee order by team_size desc;




--Q22.
--Table: Countries

create table if not exists Countries
(
    country_id int not null,
    country_name varchar(50),
    constraint pk PRIMARY KEY (country_id)
);


insert into Countries VALUES
(2,'USA'),(3,'Australia'),(7,'Peru'),(5,'China'),(8,'Morocco'),(9,'Spai
n');

select * from Countries;

--Table: Weather
create table if not exists Weather
(
    country_id int,
    weather_state int,
    day date,
    constraint pk 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'),(5,1
8,'2019-11-09'),(5,21,'2019-11-23'),(7,25,'2019-11-28'),(7,22,'2019-12-
01'),(7,20,'2019-12-02'),(8,25,'2019-11-05'),(8,27,'2019-11-15'),(8,31,
'2019-11-25'),(9,7,'2019-10-23'),(9,3,'2019-12-23');

select * from Weather;

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

```
--The type of weather is:
--● Cold if the average weather_state is less than or equal 15,
--● Hot if the average weather_state is greater than or equal to 25,
and
--● Warm otherwise.
--Return result table in any order.

select c.country_name,
                    case
                        when AVG(w.weather_state*1.0) <= 15 then 'Cold'
                        when AVG(w.weather_state*1.0) >= 25 then 'Hot'
                        else 'Warm'
                    end as weather_type
from Countries as c
inner JOIN Weather w ON c.country_id = w.country_id
where w.day between '2019-11-01' and '2019-11-30'
group by c.country_id;



--Q23.
--Table: Prices
create table if not exists Prices
(
    product_id int,
    start_date date,
    end_date date,
    price int,
    constraint pk PRIMARY KEY (product_id, start_date, end_date)
);

insert into Prices VALUES
(1,'2019-02-17','2019-02-28',5),(1,'2019-03-01','2019-03-22',20),(2,'20
19-02-01','2019-02-20',15),(2,'2019-02-21','2019-03-31',30);

select * from Prices;

--Table: UnitsSold
create table if not exists UnitsSold
(
    product_id int,
    purchase_date date,
    units int
);


insert into UnitsSold VALUES
(1,'2019-02-25',100),(1,'2019-03-01',15),(2,'2019-02-10',200),(2,'2019-
03-22',30);

select * from UnitsSold;
```

```
--Write an SQL query to find the average selling price for each
product. average_price should be rounded to 2 decimal places.
--Return the result table in any order.


SELECT a.product_id
      , round(SUM(a.units * b.price) / SUM(a.units), 2) AS
average_price
FROM UnitsSold a
      JOIN Prices b
      ON (a.product_id = b.product_id
            AND a.purchase_date >= b.start_date
            AND a.purchase_date <= b.end_date)
GROUP BY product_id;

-- select product_id, ifnull(round(sum(prices_sum) / sum(units), 2), 0)
as average_price
--     from (
--         select p.product_id as product_id, units, price * units as
prices_sum
--             from Prices p left join UnitsSold u
--             on p.product_id = u.product_id and purchase_date between
start_date and end_date
--     ) as temp
--     group by product_id;




--Q24.
--Table: Activity
create table if not exists Activity
(
    player_id int,
    device_id int,
    event_date date,
    games_played INT DEFAULT 0,
    constraint pk PRIMARY KEY (player_id, event_date)
);

INSERT into Activity values
(1,2,'2016-03-01',5),(1,2,'2016-05-02',6),(2,3,'2017-06-25',1),(3,1,'20
16-03-02',0),(3,4,'2018-07-03',5);


--Write an SQL query to report the first login date for each
player.Return the result table in any order.
select player_id,event_date as first_login,
       row_number() over(partition by player_id) as row_num
from Activity;
```

```
select
     tmp.player_id,tmp.event_date as first_login
from (select *,
        row_number() over(partition by player_id ) as row_num
     from Activity) tmp
where tmp.row_num = 1;


--Q25.
--Table: Activity
--Write an SQL query to report the device that is first logged in for
each player.
--Return the result table in any order.
select
     tmp.player_id,tmp.device_id
from (select *,
        row_number() over(partition by player_id ) as row_num
     from Activity) tmp
where tmp.row_num = 1;


--Q26.
--Table: Products
create table if not exists Products
(
    product_id int,
    product_name VARCHAR(50),
    product_category VARCHAR(50),
    constraint pk PRIMARY KEY (product_id)
);

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

select * from Products;


--Table: Orders
create table if not exists Orders
(
    product_id int,
    order_date date,
    unit int,
    constraint fk FOREIGN KEY (product_id) REFERENCES
Products(product_id)
);
```

```
insert into Orders values
(1,'2020-02-05',60),(1,'2020-02-10',70),(2,'2020-01-18',30),(2,'2020-02
-11',80),(3,'2020-02-17',2),(3,'2020-02-24',3),(4,'2020-03-01',20),(4,'
2020-03-04',30),(4,'2020-03-04',60),(5,'2020-02-25',50),(5,'2020-02-27'
,50),(5,'2020-03-01',50);

select * from Orders;

--Write an SQL query to get the names of products that have at least
100 units ordered in February 2020 and their amount.
--Return result table in any order.
select a.product_name, sum(unit) as unit
from Products a
left join Orders b
on a.product_id = b.product_id
where b.order_date between '2020-02-01' and '2020-02-29'
group by a.product_id
having sum(unit) >= 100;



--Q27.
--Table: Users
create table if not exists Users
(
    user_id int,
    name varchar(50),
    mail varchar(50),
    constraint pk PRIMARY KEY (user_id)
);


insert into Users VALUES
(1,'Winston','winston@leetcode.com'),(2,'Jonathan','jonathanisgreat'),(
3,'Annabelle','bella-@leetcode.com'),(4,'Sally','sally.come@leetcode.co
m'),(5,'Marwan','quarz#2020@le
etcode.com'),(6,'David','david69@gmail.com'),(7,'Shapiro','.shapo@leetc
ode.com');

select * from Users;

--Write an SQL query to find the users who have valid emails.
--A valid e-mail has a prefix name and a domain where:
--● The prefix name is a string that may contain letters (upper or
lower case), digits, underscore
--'_', period '.', and/or dash '-'. The prefix name must start with a
letter.
--● The domain is '@leetcode.com'.
--Return the result table in any order.

SELECT *
FROM Users
```

```
WHERE REGEXP_LIKE(mail, '^[a-zA-Z][a-zA-Z0-9\_\.\-]*@leetcode.com');


--Q28.
--Table: Customers
create table if not exists Customers
(
    customer_id int,
    name varchar(50),
    country varchar(50),
    constraint pk PRIMARY KEY (customer_id)
);


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

select * from Customers;


--Table: Product
create table if not exists Product
(
    product_id int,
    description varchar(255),
    price int,
    constraint pk PRIMARY KEY (product_id)
);

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

select * from Product;

--Table: Orders
create table if not exists Orders
(
    order_id int,
    customer_id int,
    product_id int,
    order_date DATE,
    quantity int,
    constraint pk PRIMARY KEY (order_id)
    -- constraint fk FOREIGN KEY (customer_id) REFERENCES
Customers(customer_id),
    -- constraint fk FOREIGN KEY (product_id) REFERENCES
Product(product_id)
);
```

```
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 * from Orders;


--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.
select o.customer_id, c.name
from Customers c, Product p, Orders o
where c.customer_id = o.customer_id and p.product_id = o.product_id
group by o.customer_id
having
(
    sum(case when o.order_date like '2020-06%' then o.quantity*p.price
else 0 end) >= 100
    and
    sum(case when o.order_date like '2020-07%' then o.quantity*p.price
else 0 end) >= 100
);



--Q29.
--Table: TVProgram
create table if not exists TVProgram
(
    program_date date,
    content_id int,
    channel varchar(50),
    constraint pk PRIMARY KEY (program_date, content_id)
);

insert into TVProgram VALUES ('2020-06-10
08:00',1,'LC-Channel'),('2020-05-11 12:00',2,'LC-Channel'),('2020-05-12
12:00',3,'LC-Channel'),('2020-05-13 14:00',4,'Disney Ch'),('2020-06-18
14:00',4,'Disney Ch'),('2020-07-15 16:00',5,'Disney Ch');

select * from TVProgram;

--Table: Content
create table if not exists Content
(
    content_id varchar(50),
    title varchar(50),
    Kids_content enum('Y','N'),
    content_type varchar(50),
    constraint pk 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','Mov
ies');
select * from Content;

--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.

SELECT DISTINCT title
FROM Content ctt
INNER JOIN TVProgram tv
ON ctt.content_id = tv.content_id
WHERE content_type = 'Movies'
AND Kids_content = 'Y'
AND program_date BETWEEN '2020-06-01' AND '2020-06-30';

--Q30.
--Table: NPV
create table if not exists NPV
(
    id int,
    year int,
    npv int,
    constraint pk PRIMARY KEY (id, year)
);

insert into NPV VALUES
(1,2018,100),(7,2020,30),(13,2019,40),(1,2019,113),(2,2008,121),(3,2009
,12),(11,2020,99),(7,2019,0);

select * from NPV;

--Table: Queries
create table if not exists Queries
(
    id int,
    year int,
    constraint pk PRIMARY KEY (id, year)
);

insert into Queries VALUES (1,
2019),(2,2008),(3,2009),(7,2018),(7,2019),(7,2020),(13,2019);

select * from Queries;
```

```
--Q31.

--Write an SQL query to find the npv of each query of the Queries
table.
--Return the result table in any order.
SELECT q.id, q.year, COALESCE(n.npv,0) AS npv
FROM Queries q
LEFT JOIN NPV n
ON q.id = n.id AND q.year=n.year;


--Q32.
--Table: Employees
create table if not exists Employees
(
    id int,
    name varchar(50),
    constraint pk PRIMARY KEY (id)
);

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

select * from Employees;

--Table: EmployeeUNI
create table if not exists EmployeeUNI
(
    id int,
    unique_id int,
    constraint pk PRIMARY KEY (id, unique_id)
);

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


--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.
select unique_id, name
from Employees
left join EmployeeUNI
on if (Employees.id = EmployeeUNI.id, Employees.id, null);


--Q33.
--Table: Users
create table if not exists Users
(
    id int,
    name VARCHAR(50),
    constraint pk PRIMARY KEY (id)
);

insert into Users VALUES
(1,'Alice'),(2,'Bob'),(3,'Alex'),(4,'Donald'),(7,'Lee'),(13,'Jonathan')
,(19,'Elvis');

select * from Users;

--Table: Rides
create table if not exists Rides
(
    id int,
    user_id int,
    distance int,
    constraint pk PRIMARY KEY (id),
    constraint fk FOREIGN KEY (user_id) REFERENCES Users(id)
);

insert into Rides VALUES
(1,1,120),(2,2,317),(3,3,222),(4,7,100),(5,13,312),(6,19,50),(7,7,120),
(8,19,400),(9,7,230);

select * from Rides;

--Write an SQL query to report the distance travelled by each user.
--Return the result table ordered by travelled_distance in descending
order, if two or more users travelled the same distance, order them by
their name in ascending order.
select name, sum(ifnull(distance, 0)) as travelled_distance
from Rides r
right join Users u
on r.user_id = u.id
group by name
order by 2 desc,1 asc;



--Q34.
--Table: Products
create table if not exists Products
(
    product_id int,
    product_name varchar(50),
    product_category VARCHAR(50),
    constraint pk PRIMARY KEY (product_id)
);

insert into Products VALUES (1,'Leetcode Solutions','Book'),(2,'Jewels
of
```

```
Stringology','Book'),(3,'HP','Laptop'),(4,'Lenovo','Laptop'),(5,'Leetco
de Kit','T-shirt');

select * from Products;

--Table: Orders
create table if not exists Orders
(
    product_id int,
    order_date date,
    unit int
);

insert into Orders values
(1,'2020-02-05',60),(1,'2020-02-10',70),(2,'2020-01-18',30),(2,'2020-02
-11',80),(3,'2020-02-17',2),(3,'2020-02-24',3),(4,'2020-03-01',20),(4,'
2020-03-04',30),(4,'2020-03-04',60),(5,'2020-02-25',50),(5,'2020-02-27'
,50),(5,'2020-03-01',50);

select * from Orders;

--Write an SQL query to get the names of products that have at least
100 units ordered in February 2020 and their amount.
--Return result table in any order.
select a.product_name, sum(unit) as unit
from Products a
left join Orders b
on a.product_id = b.product_id
where b.order_date between '2020-02-01' and '2020-02-29'
group by a.product_id
having sum(unit) >= 100;


--Q35.
--Table: Movies
create table if not exists Movies
(
    movie_id int,
    title varchar(50),
    constraint pk PRIMARY KEY (movie_id)
);

insert into Movies VALUES (1,'Avengers'),(2,'Frozen 2'),(3,'Joker');
select * from Movies;

--Table: Users
create table if not exists Users
(
    user_id int,
    name varchar(50),
    constraint pk PRIMARY KEY (user_id)
```

```
);

insert into Users VALUES
(1,'Daniel'),(2,'Monica'),(3,'Maria'),(4,'James');
select * from Users;

--MovieRating table:
create table if not exists MovieRating
(
    movie_id int,
    user_id int,
    rating int,
    created_at date,
    constraint pk PRIMARY KEY (movie_id, user_id)
);

insert into MovieRating VALUES
(1,1,3,'2020-01-12'),(1,2,4,'2020-02-11'),(1,3,2,'2020-02-12'),(1,4,1,'
2020-01-01'),(2,1,5,'2020-02-17'),(2,2,2,'2020-02-01'),(2,3,2,'2020-03-
01'),(3,1,3,'2020-02-22'),(3,2,4,'2020-02-25');
select * from MovieRating;

--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.

SELECT user_name AS results FROM
(
SELECT a.name AS user_name, COUNT(*) AS counts FROM MovieRating AS b
    JOIN Users AS a
    on a.user_id = b.user_id
    GROUP BY b.user_id
    ORDER BY counts DESC, user_name ASC LIMIT 1
) first_query
UNION
SELECT movie_name AS results FROM
(
SELECT c.title AS movie_name, AVG(d.rating) AS rate FROM MovieRating AS
d
    JOIN Movies AS c
    on c.movie_id = d.movie_id
    WHERE substr(d.created_at, 1, 7) = '2020-02'
    GROUP BY d.movie_id
    ORDER BY rate DESC, movie_name ASC LIMIT 1
) second_query;
```

```
--Q36.Table: Users
create table if not exists Users
(
    id int,
    name varchar(50),
    constraint pk PRIMARY KEY (id)
);

insert into Users VALUES
(1,'Alice'),(2,'Bob'),(3,'Alex'),(4,'Donald'),(7,'Lee'),(13,'Jonathan')
,(19,'Elvis');

select * from Users;


--Table: Rides
create table if not exists Rides
(
    id int,
    user_id int,
    distance int,
    constraint pk PRIMARY KEY (id),
    constraint fk FOREIGN KEY (user_id) REFERENCES Users(id)
);

insert into Rides VALUES
(1,1,120),(2,2,317),(3,3,222),(4,7,100),(5,13,312),(6,19,50),(7,7,120),
(8,19,400),(9,7,230);

select * from Rides;

--Write an SQL query to report the distance travelled by each user.
--Return the result table ordered by travelled_distance in descending
order, if two or more users
--travelled the same distance, order them by their name in ascending
order.
select name, sum(ifnull(distance, 0)) as travelled_distance
from Rides r
right join Users u
on r.user_id = u.id
group by name
order by 2 desc,1 asc;




--Q37.
--Table: Employees
create table if not exists Employees
(
    id int,
    name varchar(50),
    constraint pk PRIMARY KEY (id)
);

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

select * from Employees;

--Table: EmployeeUNI
create table if not exists EmployeeUNI
(
    id int,
    unique_id int,
    constraint pk PRIMARY KEY (id, unique_id)
);

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


--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.
select unique_id, name
from Employees
left join EmployeeUNI
on if (Employees.id = EmployeeUNI.id, Employees.id, null);


--Q38.
--Table: Departments
create table if not exists Departments
(
    id int,
    name varchar(50),
    constraint pk PRIMARY KEY (id)
);

insert into Departments VALUES (1,'Electrical
Engineering'),(7,'Computer Engineering'),(13,'Business
Administration');

select * from Departments;

--Table: Students
create table if not exists Students
(
    id int,
    name varchar(50),
    department_id int,
    constraint pk PRIMARY KEY (id)
);

insert into Students VALUES
(23,'Alice',1),(1,'Bob',7),(5,'Jennifer',13),(2,'John',14),(4,'Jasmine'
,77),(3,'Steve',74),(6,'Luis',1),(8,'Jonathan',7),(7,'Daiana',33),(11,'
Madelynn',1);

select * from Students;

--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.
select s.id, s.name
from Students s
left join Departments d
on s.department_id = d.id
where d.id is null;


--Q39.
--Table: Calls
create table if not exists Calls
(
    from_id int,
    to_id int,
    duration int
);

insert into Calls VALUES
(1,2,59),(2,1,11),(1,3,20),(3,4,100),(3,4,200),(3,4,200),(4,3,499);

select * from Calls;

--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.

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;


--Q40.
--Table: Prices
create table if not exists Prices
```

```
(
    product_id int,
    start_date date,
    end_date date,
    price int,
    constraint pk PRIMARY KEY (product_id, start_date, end_date)
);

insert into Prices VALUES (1,'2019-02-17','2019-02-28',5),
(1,'2019-03-01','2019-03-22',20), (2,'2019-02-01','2019-02-20',15),
(2,'2019-02-21','2019-03-31',30);

select * from Prices;

--UnitsSold table:
create table if not exists UnitsSold
(
    product_id int,
    purchase_date date,
    units int
);

insert into UnitsSold VALUES
(1,'2019-02-25',100),(1,'2019-03-01',15),(2,'2019-02-10',200),(2,'2019-
03-22',30);

select * from UnitsSold;


--Write an SQL query to find the average selling price for each
product. average_price should be rounded to 2 decimal places.
--Return the result table in any order.
select p.product_id,
    round(sum(p.price * u.units)/sum(u.units), 2) as average_price
from Prices p
left join UnitsSold u
on p.product_id = u.product_id and
    datediff(u.purchase_date, p.start_date) >= 0 and
    datediff(p.end_date, u.purchase_date) >= 0
group by p.product_id;



--Q41.
--Table: Warehouse
create table if not exists Warehouse
(
    name VARCHAR(50),
    product_id int,
    units int,
    constraint pk PRIMARY KEY (name, product_id)
);

insert into Warehouse VALUES
('LCHouse1',1,1),('LCHouse1',2,10),('LCHouse1',3,5),('LCHouse2',1,2),('
LCHouse2',2,2),('LCHouse3',4,1);
select * from Warehouse;

--Table: Products
create table if not exists Products
(
    product_id int,
    product_name VARCHAR(50),
    Width int,
    Length int,
    Height int,
    constraint pk 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-T-Shirt',4,10,20);

select * from Products;

--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.
select name as warehouse_name, sum(units * vol) as volume
from Warehouse w
join (select product_id, Width*Length*Height as vol
     from Products) p
on w.product_id = p.product_id
group by name;


--Q42.
--Table: Sales
create table if not exists Sales
(
    sale_date date,
    fruit enum("apples","oranges"),
    sold_num int,
    constraint pk PRIMARY KEY (sale_date, fruit)
);

insert into Sales VALUES
('2020-05-01','apples',10),('2020-05-01','oranges',8),('2020-05-02','ap
ples',15),('2020-05-02','oranges',15),('2020-05-03','apples',20),('2020
-05-03','oranges',0),('2020-05-04','apples',15),('2020-05-04','oranges'
,16);
```

```
select * from Sales;

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

select a.sale_date, (a.sold_num - b.sold_num) as diff
from Sales a left join Sales b
on a.sale_date = b.sale_date
where a.fruit = 'apples' and b.fruit = 'oranges';



--Q43.
--Table: Activity
create table if not exists Activity
(
    player_id int,
    device_id int,
    event_date date,
    games_played int,
    constraint pk 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,'20
16-03-02',0),(3,4,'2018-07-03',5);

select * from Activity;

--Write an SQL query to report the fraction of players that logged in
again on the day after the day they first logged in, rounded to 2
decimal places. In other words, you need to count the number of players
that logged in for at least two consecutive days starting from their
first login date, then divide that number by the total number of
players.
WITH CTE AS (
SELECT
player_id, min(event_date) as event_start_date
from
Activity
group by player_id )

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

```
--Q44.
--Table: Employee
create table if not exists Employee
(
    id int,
    name VARCHAR(50),
    department VARCHAR(50),
    managerId int default null,
    constraint pk PRIMARY KEY (id)
);


insert into Employee VALUES (101,'John','A',null), (102,'Dan','A',101),
(103,'James','A',101), (104,'Amy','A',101), (105,'Anne','A',101),
(106,'Ron','B',101);

select * from Employee;

--Write an SQL query to report the managers with at least five direct
reports.
--Return the result table in any order.
select
    a.name
from
    Employee a
inner join
    Employee b
on (a.id = b.managerid)
group by a.name
having count(distinct b.id) >= 5;

-- select Name from Employee
-- where Id in
-- (
--   select ManagerId from Employee
--   group by 1
--   having count(*) >= 5
-- );


--Q45.

--Table: Department
create table if not exists Department
(
    dept_id int,
    dept_name VARCHAR(50),
    constraint pk PRIMARY KEY (dept_id)
);
```

```
insert into Department VALUES
(1,'Engineering'),(2,'Science'),(3,'Law');

select * from Department;

--Table: Student
create table if not exists Student
(
    student_id int,
    student_name VARCHAR(50),
    gender VARCHAR(50),
    dept_id int,
    constraint pk PRIMARY KEY (student_id),
    constraint fk FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);

insert into Student VALUES
(1,'Jack','M',1),(2,'Jane','F',1),(3,'Mark','M',2);

select * from Student;


--Write an SQL query to report the respective department name and
number of students majoring in each department for all departments in
the Department table (even ones with no current students).
--Return the result table ordered by student_number in descending
order. In case of a tie, order them by dept_name alphabetically.
select
    a.dept_name,
    coalesce(count(student_id), 0) student_number
from
    Department a
left join
    Student b
on
    (a.dept_id = b.dept_id)
group by a.dept_name
order by student_number desc, a.dept_name asc;


--Q46.
--Table: Product
create table if not exists Product
(
    product_key int,
    constraint pk PRIMARY KEY (product_key)
);

insert into Product VALUES (5),(6);
```

```
select * from Product;

--Table: Customer
create table if not exists Customer
(
    customer_id int,
    product_key int,
    constraint fk FOREIGN KEY (product_key) REFERENCES
Product(product_key)
);


insert into Customer VALUES (1,5),(2,6),(3,5),(3,6),(1,6);

select * from Customer;

--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.
SELECT
    customer_id
FROM customer
GROUP BY customer_id
HAVING COUNT( DISTINCT product_key) = (SELECT COUNT(*) FROM product);


--Q47.
--Table: Employee
create table if not exists Employee
(
    employee_id int,
    name VARCHAR(50),
    experience_years int,
    constraint pk PRIMARY KEY (employee_id)
);

insert into Employee VALUES
(1,'Khaled',3),(2,'Ali',2),(3,'John',3),(4,'Doe',2);

select * from Employee;

--Table: Project
create table if not exists Project
(
    project_id int,
    employee_id int,
    constraint pk PRIMARY KEY (project_id, employee_id),
    constraint fk FOREIGN KEY (employee_id) REFERENCES
Employee(employee_id)

);
```

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

select * from Project;

--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.
SELECT
    project_id,
    employee_id
FROM (
    SELECT
        p.project_id,
        p.employee_id,
        DENSE_RANK() OVER(PARTITION BY p.project_id ORDER BY
e.experience_years DESC) as rnk
    FROM Project as p JOIN Employee as e
    ON p.employee_id = e.employee_id
    ) x
WHERE rnk = 1;


--Q48.
--Table: Books
create table if not exists Books
(
    book_id int,
    name VARCHAR(50),
    available_from date,
    constraint pk PRIMARY KEY (book_id)
);

insert into Books VALUES (1,'"Kalila And Demna"','2010-01-01'),(2,'"28
Letters"','2012-05-12'),(3,'"The Hobbit"','2019-06-10'),(4,'"13 Reasons
Why"','2010-01-01'),(5,'"The Hunger Games"','2008-09-21');

select * from Books;

--Table: Orders
create table if not exists Orders
(
    order_id int,
    book_id int,
    quantity int,
    dispatch_date date,
    constraint pk PRIMARY KEY (order_id),
    constraint fk FOREIGN KEY (book_id) REFERENCES Books(book_id)
);
```

```
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 * from Orders;
--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.

select Books.book_id, name from Books join Orders
    on Books.book_id = Orders.book_id
    where available_from < '2019-05-23'
    and dispatch_date between '2018-06-23' and '2019-06-23'
    group by Books.book_id
    having sum(quantity) < 10
    union
select book_id, name from Books
    where available_from < '2019-05-23'
    and book_id not in (
        select distinct book_id from Orders where dispatch_date between
'2018-06-23' and '2019-06-23'
    );


--Q49.
--Table: Enrollments
create table if not exists Enrollments
(
    student_id int,
    course_id int,
    grade int,
    constraint pk PRIMARY KEY (student_id, course_id)
);

insert into Enrollments VALUES
(2,2,95),(2,3,95),(1,1,90),(1,2,99),(3,1,80),(3,2,75),(3,3,82);

select * from Enrollments;

--Write a SQL query to find the highest grade with its corresponding
course for each student. In case of a tie, you should find the course
with the smallest course_id.
--Return the result table ordered by student_id in ascending order.
select e.student_id, e.course_id, e.grade
from (
  select *, row_number() over (partition by student_id order by grade
desc) rn
  from Enrollments
) e
where e.rn = 1;


--Q50.
--Table: Teams

--Players table:
create table if not exists Players
(
    player_id int,
    group_id int,
    constraint pk PRIMARY KEY (player_id)
);

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

select * from Players;

--Table: Matches
create table if not exists Matches
(
    match_id int,
    first_player int,
    second_player int,
    first_score int,
    second_score int,
    constraint pk PRIMARY KEY (match_id)
);

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

select * from Matches;

--Write an SQL query to find the winner in each group.
--Return the result table in any order.
select group_id,player_id from
(select group_id,player_id,sum((
    case when player_id = first_player then first_score
         when player_id = second_player then second_score
         end
)) as totalScores
from Players p,Matches m
where p.player_id = m.first_player
or p.player_id = m.second_player
group by group_id,player_id
order by group_id,totalScores desc,player_id) as temp
group by group_id
order by group_id,totalScores desc,player_id;
```