Table of Contents

Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. 4

Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. 4

Q3. Query all columns (attributes) for every row in the CITY table. 4

Q4. Query all columns for a city in CITY with the ID 1661. 4

Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is 4

JPN. 4

Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is 4

JPN. 4

Q7. Query a list of CITY and STATE FROM the STATION table. 5

Q8. Query a list of CITY names FROM STATION for cities that have an even ID number. Print the results 5

Q9. Find the difference between the total number of CITY entries in the table and the number of 5

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. 5

Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) FROM STATION. Your result 6

Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) FROM STATION. Your result cannot 6

Q13. Query the list of CITY names FROM STATION that do not start with vowels. Your result cannot contain duplicates. 6

Q14. Query the list of CITY names FROM STATION that do not end with vowels. Your result cannot contain duplicates. 6

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. 6

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. 6

Q17. Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive. 7

Q18. Write an SQL query to find all the authors that viewed at least one of their own articles. 8

Q19. Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places. 9

Q20. Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.. 9

Q21 Write an SQL query to find the team size of each of the employees. 10

Q**22.** Write an SQL query to find the type of weather in each country for November 2019..... 11

Q**23.** Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places. 12

Q 24. Write an SQL query to report the first login date for each player. 13

Q25. Write an SQL query to report the device that is first logged in for each player.. 13

Q26. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount. 14

Q27. Write an SQL query to find the users who have valid emails. 15

Q28. Write an SQL query to report the customer_id and customer_name of customers who have spent at least $100 in each month of June and July 2020. 16

Q29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020. 17

Q30. Write an SQL query to find the npv of each query of the Queries table. 19

Q.31..... 20

Q32. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null. 20

Q33. Write an SQL query to report the distance travelled by each user. 21

Q34. Repeated from question 29 21

Q35. Write an SQL query to: 22

● 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. 22

● Find the movie name with the highest average rating in February 2020. In case of a tie, return the lexicographically smaller movie name. 22

Q36..... 23

Q37...... 23

Q38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist. 24

Q39. Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2. 25

Q40. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places. 25

Q41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse. 26

Q42. Write an SQL query to report the difference between the number of apples and oranges sold each day. 27

Q43. Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players. 27

Q44. Write an SQL query to report the managers with at least five direct reports. ... 28

Q45. Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students)..... 29

Q46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table. 30

Q47. Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years. 31

Q48. Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23..... 32

Q49. Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course_id. 33

Q50. Write an SQL query to find the winner in each group. Return the result table in any order. 34

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

Solution:

```
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.
Solution:

```
SELECT NAME

FROM CITY

WHERE COUNTRYCODE = 'USA' and POPULATION > 120000;
```

Q3. Query all columns (attributes) for every row in the CITY table.
Solution: SELECT * FROM CITY;

Q4. Query all columns for a city in CITY with the ID 1661.
Solution: SELECT *

FROM CITY

WHERE ID = 1661;

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

FROM CITY

WHERE COUNTRYCODE = 'JPN';

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

```
SELECT NAME

FROM CITY
```

```
WHERE COUNTRYCODE  = 'JPN';
```

Q7. Query a list of CITY and STATE FROM the STATION table.
Solution:

```
SELECT CITY, STATE
FROM STATION;
```

Q8. Query a list of CITY names FROM STATION for cities that have an even ID number. Print the results
in any order, but exclude duplicates FROM the answer.

Solution:

```
SELECT DISTINCT CITY
FROM STATION
WHERE MOD(ID,2)=0;
```

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

Solution:

```
SELECT COUNT(CITY) - COUNT(DISTINCT(CITY)) as Difference
FROM STATION;
```

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

```
SELECT CITY, length(CITY)
FROM STATION
ORDER BY length(CITY), CITY limit 1
```

```
SELECT CITY, length(CITY)
FROM STATION
ORDER BY length(CITY) desc, CITY limit 1;
```

Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) FROM STATION. Your result
cannot contain duplicates.

Solution:

```
SELECT DISTINCT(CITY) FROM STATION WHERE city REGEXP '^[AEIOUaeiou]';
```

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

Solution:

```
SELECT DISTINCT(CITY) FROM STATION WHERE city REGEXP '[aeiouAEIOU]$';
```

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

```
SELECT DISTINCT(CITY) FROM STATION WHERE city not REGEXP '^[AEIOUaeiou]';
```

```
SELECT DISTINCT (city) FROM station WHERE city REGEXP '^[^aeiouAEIOU]';
```

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

```
SELECT DISTINCT(CITY) FROM STATION WHERE city not REGEXP '[aeiouAEIOU]$';
```

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

```
SELECT DISTINCT(CITY) FROM STATION WHERE city REGEXP '^[^aeiouAEIOU]' OR' city
REGEXP '[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.
Solution:

```
SELECT DISTINCT city FROM station WHERE city REGEXP '^[^aeiouAEIOU]' AND city REGEXP
'[^aeiouAEIOU]$';
```

Q17. Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.
Return the result table in any order.

Solution:

```
CREATE TABLE  PRODUCT
        (
                PRODUCT_ID INT PRIMARY KEY,
                PRODUCT_NAME VARCHAR(25),
                UNIT_PRICE INT
        );

CREATE TABLE SALES
        (
                SELLER_ID INT,
                PRODUCT_ID INT,
                BUYER_ID INT,
                SALES_DATE DATE,
                QUANTITY INT,
                PRICE INT,
                FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT(PRODUCT_ID)
        );


INSERT INTO PRODUCT VALUES (1, "S8", 1000);
INSERT INTO PRODUCT VALUES (2, "G4", 800);
INSERT INTO PRODUCT VALUES (3, "IPHONE", 1400);

INSERT INTO SALES VALUES (1, 1, 1, "2019-01-21", 2, 2000);
INSERT INTO SALES VALUES (2, 2, 2, "2019-02-17", 2, 800);
INSERT INTO SALES VALUES (3, 3, 3, "2019-06-02", 2, 900);
INSERT INTO SALES VALUES (4, 4, 4, "2019-05-13", 2, 2800);
```

```
SELECT PRODUCT_NAME
    FROM PRODUCT
    WHERE PRODUCT_ID NOT IN
        (SELECT PRODUCT_ID FROM SALES WHERE SALES_DATE BETWEEN "2019-01-31" AND "2019-03-31")
        AND
        PRODUCT_ID IN
        (SELECT PRODUCT_ID FROM SALES WHERE SALES_DATE < "2019-03-31");
```

Q18. Write an SQL query to find all the authors that viewed at least one of their own articles.
Return the result table sorted by id in ascending order.

Solution:

```
CREATE TABLE VIEWS
(
ARITICLE_ID INT,
AUTHOR_ID INT,
VIEWER_ID INT,
VIEW_DATE DATE
);
INSERT INTO VIEWS VALUE (1, 3, 5, "2019-08-01");
INSERT INTO VIEWS VALUE (1, 3, 6, "2019-08-01");
INSERT INTO VIEWS VALUE (2, 7, 7, "2019-08-01");
INSERT INTO VIEWS VALUE (1, 3, 5, "2019-08-01");
INSERT INTO VIEWS VALUE (1, 3, 5, "2019-08-01");
INSERT INTO VIEWS VALUE (1, 3, 5, "2019-08-01");
INSERT INTO VIEWS VALUE (1, 3, 5, "2019-08-01");

SELECT AUTHOR_ID
FROM VIEWS
WHERE AUTHOR_ID = VIEWER_ID
GROUP BY AUTHOR_ID
ORDER BY AUTHOR_ID;
```

Q19. Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

```
CREATE TABLE DELIVERY
(
DELIVERY_ID INT,
CUSTOMER_ID INT,
ORDER_DATE DATE,
CUSTOMER_PREF_DELIVERY_DATE DATE
);

INSERT INTO DELIVERY VALUE (1, 1, "2019-08-01", "2019-08-02");
INSERT INTO DELIVERY VALUE (1, 1, "2019-08-02", "2019-08-02");
INSERT INTO DELIVERY VALUE (1, 1, "2019-08-11", "2019-08-11");
INSERT INTO DELIVERY VALUE (1, 1, "2019-08-24", "2019-08-26");
INSERT INTO DELIVERY VALUE (1, 1, "2019-08-21", "2019-08-22");
INSERT INTO DELIVERY VALUE (1, 1, "2019-08-11", "2019-08-13");
```

Solution:

```
SELECT ROUND(SUM(ORDER_DATE = CUSTOMER_PREF_DELIVERY_DATE)/ COUNT(*) *
100,2) AS IMMEDIATE_DELIVERY_PERCENTAGE FROM DELIVERY ;
```

Q20. Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.
Return the result table ordered by ctr in descending order and by ad_id in ascending order in case of a
tie.

Solution:

```
CREATE TABLE ADS
(
AD_ID INT,
USER_ID INT,
ACTION ENUM (' CLICKED', ' VIEWED', ' IGNORED')
);
```

```
INSERT INTO ADS VALUES (1, 1, ' CLICKED'),

INSERT INTO ADS VALUES (2, 2, ' CLICKED'),

INSERT INTO ADS VALUES (3, 3, 'VIEWED),

INSERT INTO ADS VALUES (5, 5, 'IGNORED'),

INSERT INTO ADS VALUES (1, 7, ' IGNORED'),

INSERT INTO ADS VALUES (2, 7, 'VIEWED'),

INSERT INTO ADS VALUES (3, 5, ' CLICKED'),

INSERT INTO ADS VALUES (1, 4, 'VIEWED'),

INSERT INTO ADS VALUES (2, 11, 'VIEWED'),

INSERT INTO ADS VALUES (1, 2, ' CLICKED');


SELECT DISTINCT AD_ID,


IFNULL(SUM(ACTION="CLICKED")/(SUM(ACTION="CLICKED")+ SUM(ACTION="VIEWED"))*100,
0) AS CTR

FROM ADS GROUP BY AD_ID ORDER BY CTR DESC;
```

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

```
CREATE TABLE

(

EMPLOYEE_ID INT PRIMARY KEY,

TEAM_ID INT

);


INSERT INTO EMPLOYEE VALUES (1, 8), (2, 8), (3, 8), (4, 7), (5, 9), (6, 9);


SELECT E1.EMPLOYEE_ID, COUNT(E2.EMPLOYEE_ID) AS TEAM_SIZE

FROM

EMPLOYEE E1 INNER JOIN EMPLOYEE E2

ON E1.TEAM_ID=E2.TEAM_ID
```

```
GROUP BY E1.EMPLOYEE_ID, E2.TEAM_ID;

SELECT EMPLOYEE_ID, COUNT(EMPLOYEE_ID)

OVER(PARTITION BY TEAM_ID ORDER BY TEAM_ID) AS TEAM_SIZE

FROM EMPLOYEE;
```

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.

Solution:

```
CREATE TABLE COUNTRIES

(

COUNTRY_ID INT PRIMARY KEY,

COUNTRY_NAME VARCHAR(30)

);


CREATE TABLE WEATHER

(

COUTRY_ID INT,

WEATHER_STATE INT,

DAY DATE

);


INSERT INTO COUNTRIES VALUES (2, 'USA'), (3, 'AUSTRALIA'), (7, 'PERU'), (5, 'CHINA'), (8,
'MOROCCO'), (9, 'SPAIN');


INSERT INTO WHEATHER VALUES (2, 15, '2019-11-01'), (2, 12, '2019-10-28'), (2, 12, '2019-10-27'),
(3, -2, '2019-11-10'), (3, 0, '2019-11-11'), (3, 3, '2019-11-12'), (5, 16, '2019-11-07'), (5, 18, '2019-11-
09'), (5, 21, '2019-11-23'), (7, 25, '2019-11-28'), (7, 22, '2019-12-01'), (7, 20, '2019-12-02'), (8, 25,
'2019-11-05'), (8, 27, '2019-11-15'), (8, 31, '2019-11-25'), (9, 7, '2019-10-23'), (9, 3, '2019-12-23');


SELECT COUNTRY_NAME, CASE

WHEN AVG(WEATHER_STATE) <=15 THEN "COLD"
```

```
WHEN AVG(WEATHER_STATE) >= 25 THEN "HOT"
ELSE "WARM" END AS WEATHER_TYPE
FROM COUNTRIES
INNER JOIN WEATHER
ON COUNTRIES.COUNTRY_ID = WEATHER.COUNTRY_ID
WHERE MONTH(DAY) = '11'
GROUP BY 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.
Return the result table in any order.

Solution:

```
CREATE TABLE PRICES
(
PRODUCT_ID INT,
START_DATE DATE,
END_DATE DATE,
PRICE INT,
PRIMARY KEY (PRODUCT_ID, START_DATE, END_DATE)
);

CREATE TABLE UNITS_SOLD
(
PRODUCT_ID INT,
PURCHASE_DATE DATE,
UNITS INT
);

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);

INSERT INTO UNITS_SOLD VALUES (1, '2019-02-25', 100), (1, '2019-03-01', 15), (2, '2019-02-10', 200), (2, '2019-03-22', 30);
```

```
SELECT PRODUCT_ID, SUM(SUM_PRICES)/SUM(UNITS) AS AVG_PRICE
FROM
(SELECT P.PRODUCT_ID, P.PRICE, U.UNITS, PRICE * UNITS AS SUM_PRICES
FROM PRICES P
LEFT JOIN UNITS_SOLD U
ON P.PRODUCT_ID = U.PRODUCT_ID AND U.PURCHASE_DATE BETWEEN P.START_DATE
AND P.END_DATE) AS TEMP_PRICES
GROUP BY 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,
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, '2016-07-03', 5);

SELECT PLAYER_ID,
MIN(EVENT_DATE) AS FIRST_LOGIN
FROM
ACTIVITY
GROUP BY PLAYER_ID;
```

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

Solution:

```
CREATE TABLE ACTIVITIES
(
PLAYER_ID INT, DEVICE_ID INT,
EVENT_DATE DATE,
GAMES_PLAYED INT,
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, '2016-07-03', 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) AS RN
FROM ACTIVITIES) A
WHERE A.RN=1;
```

Q26. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

Solution:

```
CREATE TABLE PRODUCTS
(
PRODUCT_ID INT PRIMARY KEY,
PRODUCT_NAME VARCHAR (30),
PRODUCT_CATEGORY VARCHAR (30)
);

CREATE TABLE ORDERS
(
PRODUCT_ID INT,
ORDER_DATE DATE,
```

```
UNIT INT
);

INSERT INTO PRODUCTS VALUES
(1,'LEETCODE SOLUTIONS','BOOK'), (2,'JEWELS OF STRINGOLOGY','BOOK'),
(3,'HP','LAPTOP'), (4,'LENOVO','LAPTOP'), (5,'LEETCODE KIT', 'T-SHIRT');

INSERT INTO ORDERS VALUES
(1,'2020-02-05',60), (1,'2020-02-10',70), (2,'2020-01-18',30), (2,'2020-02-11',80),
(3,'2020-02-17',2), (3,'2020-02-24',3), (4,'2020-03-01',20), (4,'2020-03-04',30),
(4,'2020-03-04',60), (5,'2020-02-25',50), (5,'2020-02-27',50), (5,'2020-03-01',50);
```

Solution:

```
SELECT P.PRODUCT_NAME, O.TOTAL_UNITS
FROM PRODUCTS P
JOIN
(SELECT PRODUCT_ID, SUM(UNIT) AS TOTAL_UNITS
FROM ORDERS
WHERE ORDER_DATE BETWEEN '2020-02-01' AND '2020-02-28'
GROUP BY PRODUCT_ID) O
ON P.PRODUCT_ID = O.PRODUCT_ID
WHERE O.TOTAL_UNITS >= 100;
```

Q27. 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'.

Solution:

```
CREATE TABLE USERS
(
```

```
USER_ID INT PRIMARY KEY,
NAME VARCHAR (30),
MAIL VARCHAR (50)
);

INSERT INTO USERS VALUES
(1, 'WINSTON', 'WINSTON@LEETCODE.COM'), (2, 'JONATHAN', 'JONATHANISGREAT'),
(3, 'ANNABELLE', 'BELLA-@LEETCODE.COM'), (4, 'SALLY', 'SALLY.COME@LEETCODE.COM'),
(5, 'MARWAN', 'QUARZ#2020@LEETCODE.COM'), (6, 'DAVID', 'DAVID69@GMAIL.COM'),
(7, 'SHAPIRO', '.0SHAPO@LEETCODE.COM');

SELECT * FROM USERS WHERE MAIL REGEXP '^[A-ZA-Z]+[A-ZA-Z0-9_\\./\\-
]{0,}@LEETCODE.COM$' ORDER BY USER_ID;
```

Q28. Write an SQL query to report the customer_id and customer_name of customers who have spent at least $100 in each month of June and July 2020.
Solution:

```
CREATE TABLE CUSTOMER
(
CUSTOMER_ID INT PRIMARY KEY,
NAME VARCHAR (30),
COUNTRY VARCHAR (30)
);

CREATE TABLE PRODUCTS1
(
PRODUCT_ID INT PRIMARY KEY,
DESCRIPTION VARCHAR (50),
PRICE INT
);

CREATE TABLE ORDERS1
(
ORDER_ID INT PRIMARY KEY,
```

```
CUSTOMER_ID INT,
PRODUCT_ID INT,
ORDER_DATE DATE,
QUANTITY INT
);

INSERT INTO CUSTOMER VALUES (1,'WINSTON','USA'), (2,'JONATHAN','PERU'),
(3,'MOUSTAFA','EGYPT');

INSERT INTO PRODUCT VALUES (10, 'LC PHONE', 300), (20, 'LC T-SHIRT', 10),
(30, 'LC BOOK', 45), (40, 'LC KEYCHAIN', 2);

INSERT INTO ORDERS VALUES (1, 1, 10, '2020-06-10', 1), (2, 1, 20, '2020-07-01', 1),
(3, 1, 30, '2020-07-08', 2), (4, 2, 10, '2020-06-15', 2), (5, 2, 40, '2020-07-01', 10),
(6, 3, 20, '2020-06-24', 2), (7, 3, 30, '2020-06-25', 2), (9, 3, 30, '2020-05-08', 2);

SELECT C.CUSTOMER_ID, C.NAME
FROM
CUSTOMER C
JOIN ORDERS1 O ON C.CUSTOMER_ID = O.CUSTOMER_ID
JOIN PRODUCTS1 P ON O.PRODUCT_ID = P.PRODUCT_ID
GROUP BY C.CUSTOMER_ID, C.NAME
HAVING
SUM(CASE WHEN MONTH(O.ORDER_DATE) = '06' THEN P.PRICE * O.QUANTITY ELSE 0
END)>=100 AND
SUM(CASE WHEN MONTH(O.ORDER_DATE) = '07' THEN P.PRICE * O.QUANTITY ELSE 0
END)>=100;
```

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

Return the result table in any order.

Solution:

```
CREATE TABLE TVPROGRAM
(
PROGRAM_DATE DATE,
CONTENT_ID INT,
CHANNEL VARCHAR (30),
PRIMARY KEY (PROGRAM_DATE, CONTENT_ID)
);
CREATE TABLE CONTENT
(
CONTENT_ID INT PRIMARY KEY,
TITLE VARCHAR (20),
KIDS_CONTENT ENUM ('Y','N'),
CONTENT_TYPE VARCHAR (20)
);

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');

INSERT INTO CONTENT VALUES
(1, 'LEETCODE_MOVIE', 'N', 'MOVIES'), (2, 'ALG. FOR KIDS', 'Y', 'SERIES'),
(3, 'DATABASE SOLS', 'N', 'SERIES'), (4, 'ALADDIN', 'Y', 'MOVIES'),
(5, 'CINDERELLA', 'Y', 'MOVIES');



SELECT DISTINCT TITLE
FROM CONTENT C
JOIN
TVPROGRAM T ON T.CONTENT_ID = C.CONTENT_ID
WHERE KIDS_CONTENT= 'Y'
AND CONTENT_TYPE = 'MOVIES'
```

```
AND MONTH(PROGRAM_DATE) = '06';

SELECT DISTINCT TITLE
FROM CONTENT
JOIN
TVPROGRAM USING(CONTENT_ID)
WHERE KIDS_CONTENT = 'Y'
AND CONTENT_TYPE = 'MOVIES'
AND MONTH(PROGRAM_DATE) = '06';
```

Q30. Write an SQL query to find the npv of each query of the Queries table.
Return the result table in any order.

Solution:

```
CREATE TABLE NPV
(ID INT,
YEAR INT,
NPV INT,
PRIMARY KEY (ID, YEAR)
);

CREATE TABLE QUERIES
(ID INT,
YEAR INT,
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 ID, YEAR, IFNULL(NPV, 0) AS NPV
```

```
FROM
QUERIES Q
LEFT JOIN NPV N
USING (ID, YEAR);
```

Q.31 Repeated from question 30.

Q32. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

Solution:

```
CREATE TABLE EMPLOYEES
(
ID INT PRIMARY KEY,
NAME VARCHAR (30)
);

CREATE TABLE EMPLOYEEUNI
(ID INT,
UNIQUE_ID INT,
PRIMARY KEY (ID, UNIQUE_ID)
);

INSERT INTO EMPLOYEES VALUES
(1, 'ALICE'), (7, 'BOB'), (11,'MEIR'), (90, 'WINSTON'), (3, 'JONATHAN');

INSERT INTO EMPLOYEEUNI VALUES
(3,1), (11,2), (90,3);

SELECT UNIQUE_ID, NAME
FROM EMPLOYEES
LEFT JOIN EMPLOYEEUNI
ON EMPLOYEES.ID = EMPLOYEEUNI.ID;
```

Q33. 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 PRIMARY KEY,
NAME VARCHAR (30)
);

CREATE TABLE RIDES
(
ID INT PRIMARY KEY,
USER_ID INT,
DISTANCE INT
);

INSERT INTO USERS VALUES (1, 'ALICE'), (2, 'BOB'), (3, 'ALEX'),
(4, 'DONALD'), (7, 'LEE'), (13,'JONATHAN'), (19,'ELVIS');

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

SELECT NAME, SUM(IFNULL(DISTANCE,0)) AS DISTANCE_TRAVELED
FROM RIDES R
RIGHT JOIN USERS1 U
ON R.USER_ID = U.ID
GROUP BY NAME
ORDER BY DISTANCE DESC, NAME ASC;
```

Q34. Repeated from question 29

Q35. Write an SQL query to:

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

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

Solution:

```
CREATE TABLE MOVIES (
MOVIE_ID INT PRIMARY KEY,
TITLE VARCHAR (30)
);

CREATE TABLE USERS2
(
USER_ID INT PRIMARY KEY,
NAME VARCHAR(30)
);

CREATE TABLE MOVIERATING
(
MOVIE_ID INT,
USER_ID INT,
RATING INT,
CREATED_AT DATE,
PRIMARY KEY (MOVIE_ID, USER_ID)
);

INSERT INTO MOVIES VALUES
(1,'AVENGERS'), (2,'FROZEN 2'), (3,'JOKER');

INSERT INTO USERS2 VALUES
(1, 'DANIEL'), (2, 'MONICA'), (3, 'MARIA'), (4, 'JAMES');

INSERT INTO MOVIERATING VALUES (1, 1, 3, '2020-01-12'), (1, 2, 4, '2020-02-11'),
```

```
(1, 3, 2, '2020-02-12'), (1, 4, 1, '2020-01-01'), (2, 1, 5, '2020-02-17'), (2, 2, 2, '2020-02-01'), (2, 3, 2, '2020-03-01'), (3, 1, 3, '2020-02-22'), (3, 2, 4, '2020-02-25');

ELECT USER AS MAX_REVIEWS_BY FROM

(

SELECT NAME AS USER, COUNT(*) AS NO_OF_RATINGS

FROM MOVIERATING M

JOIN USERS2 U

USING (USER_ID)

GROUP BY M.USER_ID

ORDER BY NO_OF_RATINGS DESC, USER ASC LIMIT 1

) AS GREATEST_REVIEWS


UNION

(

SELECT TITLE FROM

(

SELECT TITLE, AVG(RATING) AVG_RATING FROM MOVIERATING MR

JOIN MOVIES MV

USING (MOVIE_ID)

WHERE MONTH(CREATED_AT) = '02'

GROUP BY MOVIE_ID

ORDER BY AVG_RATING DESC, TITLE ASC LIMIT 1) AS HIGHEST_RATED_MOVIES

);
```

Q36.

Solution:

Repeated from question 31

Q37.

Solution:

Repeated from question 32

Q38. 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.

Solution:

```
CREATE TABLE DEPARTMENTS (
ID INT PRIMARY KEY,
NAME VARCHAR (30)
);

CREATE TABLE STUDENTS (ID INT PRIMARY KEY,
NAME VARCHAR (30),
DEPARTMENT_ID INT
);

INSERT INTO DEPARTMENTS VALUES
(1, 'ELECTRICAL ENGINEERING'), (7, 'COMPUTER ENGINEERING'),
(13, 'BUSINESS ADMINISTRATION');

INSERT INTO STUDENTS VALUES
(23, 'ALICE', 1), (1, 'BOB', 7), (5, 'JENNIFER', 13), (2, 'JOHN', 14), (4, 'JASMINE', 77),
(3, 'STEVE', 74), (6, 'LUIS', 1), (8, 'JONATHAN', 7), (7, 'DAIANA', 33), (11, 'MADELYNN', 1);

SELECT S.ID, S.NAME
FROM STUDENTS S
LEFT JOIN DEPARTMENTS D
ON S.DEPARTMENT_ID = D.ID
WHERE D.ID IS NULL;
```

Q39. Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2.
Return the result table in any order.

Solution:

```
CREATE TABLE CALLS
(
FROM_ID INT,
TO_ID INT,
DURATION INT
);

INSERT INTO CALLS VALUES
(1, 2, 59), (2, 1, 11), (1, 3, 20), (3, 4, 100), (3, 4, 200), (3, 4, 200), (4, 3, 499);

SELECT LEAST(FROM_ID, TO_ID) AS PERSON1, GREATEST(FROM_ID, TO_ID) AS PERSON2,
COUNT(DURATION), SUM(DURATION)
FROM
CALLS
GROUP BY PERSON1, PERSON2;
```

Q40. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places.
Return the result table in any order.

Repeated from question 23

Q41. 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.

Solution:

```
CREATE TABLE WAREHOUSE
(
NAME VARCHAR (30),
PRODUCT_ID INT,
UNITS INT,
PRIMARY KEY (NAME,PRODUCT_ID)
);

CREATE TABLE PRODUCTS
(
PRODUCT_ID INT PRIMARY KEY,
PRODUCT_NAME VARCHAR(30),
WIDTH INT,
LENGTH INT,
HEIGHT INT
);

INSERT INTO WAREHOUSE VALUES  ('LCHOUSE1',1,1), ('LCHOUSE1',2,10), ('LCHOUSE1',3,5),
('LCHOUSE2',1,2), ('LCHOUSE2',2,2), ('LCHOUSE3',4,1);

INSERT INTO PRODUCTS VALUE (1, 'LC-TV', 5, 50, 40), (2, 'LC-KEYCHAIN', 5, 5, 5),
(3, 'LC-PHONE', 2, 10, 10), (4, 'LC-T-SHIRT', 4, 10, 20);

SELECT NAME AS WAREHOUSE, SUM(UNITS * VOL) AS VOLUME
FROM WAREHOUSE W
JOIN
(SELECT PRODUCT_ID, WIDTH* LENGTH* HEIGHT AS VOL FROM PRODUCTS2 P)
PRODUCSTS2
USING (PRODUCT_ID)
GROUP BY NAME;
```

Q42. Write an SQL query to report the difference between the number of apples and oranges sold each day.

Return the result table ordered by sale_date.

Solution:

```
CREATE TABLE SALES
(
SALE_DATE DATE,
FRUIT ENUM('APPLES', 'ORANGES'),
SOLD_NUM INT,
PRIMARY KEY (SALE_DATE,FRUIT)
);

INSERT INTO SALES VALUES
('2020-05-01', 'APPLES', 10), ('2020-05-01', 'ORANGES', 8), ('2020-05-02', 'APPLES', 15),
('2020-05-02', 'ORANGES', 15), ('2020-05-03', 'APPLES', 20), ('2020-05-03', 'ORANGES', 0),
('2020-05-04', 'APPLES', 15), ('2020-05-04', 'ORANGES', 1);

SELECT DATE(SALE_DATE) AS SALE_DATE,
SUM(CASE
WHEN FRUIT = 'APPLES' THEN SOLD_NUM
WHEN FRUIT = 'ORANGES' THEN -SOLD_NUM END) AS DIFF
FROM SALES1
GROUP BY SALE_DATE ORDER BY SALE_DATE;
```

Q43. Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

Solution:

```
CREATE TABLE ACTIVITY
(
PLAYER_ID INT,
DEVICE_ID INT,
```

```
EVENT_DATE DATE ,
GAMES_PLAYED INT,
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, '2016-07-03', 5);

SELECT ROUND(IFNULL((SELECT COUNT(DISTINCT A.PLAYER_ID)
FROM ACTIVITY1 AS A
JOIN ACTIVITY1 AS B
ON A.PLAYER_ID = B.PLAYER_ID AND DATEDIFF(B.EVENT_DATE, A.EVENT_DATE) = 1
WHERE A.EVENT_DATE = (SELECT MIN(EVENT_DATE)
FROM ACTIVITY1
WHERE PLAYER_ID = A.PLAYER_ID))/ (SELECT COUNT(DISTINCT PLAYER_ID) FROM
ACTIVITY1),0),2) AS FRACION;
```

Q44. Write an SQL query to report the managers with at least five direct reports.
Return the result table in any order.

Solution:

```
CREATE TABLE EMPLOYEE2
(
ID INT PRIMARY KEY,
NAME VARCHAR(30),
DEPARTMENT VARCHAR(30),
MANAGERID INT
);

INSERT INTO EMPLOYEE VALUES
(101, 'JOHN', 'A', NULL), (102, 'DAN', 'A', 101), (103, 'JAMES', 'A', 101), (104, 'AMY', 'A', 101),
(105, 'ANNE', 'A', 101), (106, 'RON', 'B', 101);

SELECT NAME
FROM EMPLOYEE2
```

```
WHERE ID IN
(
SELECT MANAGERID

FROM EMPLOYEE2
GROUP BY MANAGERID
HAVING (COUNT(DISTINCT ID)) >= 5);
```

Q45. Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students).
Return the result table ordered by student_number in descending order. In case of a tie, order them by dept_name alphabetically.

Solution:

```
CREATE TABLE DEPARTMENT
(
DEPT_ID INT PRIMARY KEY,
DEPT_NAME VARCHAR(30)
);

CREATE TABLE STUDENT
(
STUDENT_ID INT PRIMARY KEY,
STUDENT_NAME VARCHAR(30),
GENDER VARCHAR(30),
DEPT_ID INT,
FOREIGN KEY (DEPT_ID) REFERENCES DEPARTMENT(DEPT_ID)
);

INSERT INTO DEPARTMENT VALUES
(1,'ENGINEERING'), (2,'SCIENCE'), (3,'LAW');

INSERT INTO STUDENT VALUES (1, 'JACK', 'M', 1), (2, 'JANE', 'F', 1), (3, 'MARK', 'M', 2);

SELECT D.DEPT_NAME, COALESCE(COUNT(STUDENT_ID),0) NO_OF_STUDENTS
```

```
FROM DEPARTMENT D
LEFT JOIN STUDENT S
USING (DEPT_ID)
GROUP BY D.DEPT_NAME
ORDER BY NO_OF_STUDENTS DESC, D.DEPT_NAME ASC;
```

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

Solution:

```
CREATE TABLE CUSTOMER
(
CUSTOMER_ID INT,
PRODUCT_KEY INT,
FOREIGN KEY (PRODUCT_KEY) REFERENCES PRODUCT(PRODUCT_KEY)
);

CREATE TABLE PRODUCT
(
PRODUCT_KEY INT PRIMARY KEY);

INSERT INTO PRODUCT VALUES
(5), (6);

INSERT INTO CUSTOMER VALUES
(1,5), (2,6), (3,5), (3,6), (1,6);

SELECT CUSTOMER_ID
FROM CUSTOMER1
GROUP BY CUSTOMER_ID
HAVING COUNT(DISTINCT(PRODUCT_KEY)) = (SELECT COUNT(*) FROM PRODUCT3);
```

Q47. Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years.
Return the result table in any order.

Solution:

```
CREATE TABLE EMPLOYEE3
(
EMPLOYEE_ID INT PRIMARY KEY,
NAME VARCHAR(30),
EXPERIENCE_YEARS INT
);

CREATE TABLE PROJECT
(
PROJECT_ID INT ,
EMPLOYEE_ID INT,
PRIMARY KEY (PROJECT_ID, EMPLOYEE_ID),
FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEE3(EMPLOYEE_ID)
);

INSERT INTO EMPLOYEE VALUES
(1,'KHALED',3), (2,'ALI',2), (3,'JOHN',3), (4,'DOE',2);

INSERT INTO PROJECT VALUES
(1,1), (1,2), (1,3), (2,1), (2,4);

SELECT A.PROJECT_ID , A.EMPLOYEE_ID FROM
(
SELECT  A.PROJECT_ID , A.EMPLOYEE_ID ,RANK() OVER ( PARTITION BY PROJECT_ID
ORDER BY EXPERIENCE_YEARS DESC) AS RNK
FROM PROJECT A INNER JOIN EMPLOYEE3 B ON A.EMPLOYEE_ID =B.EMPLOYEE_ID
) A
WHERE RNK = 1;
```

Q48. Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23.

Return the result table in any order.

Solution:

```
CREATE TABLE BOOKS
(
BOOK_ID INT,
NAME VARCHAR(20),
AVAILABLE_FROM DATE
);

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');

CREATE TABLE ORDERS
(
ORDER_ID INT,
BOOK_ID INT,
QUANTITY INT,
DISPATCH_DATE DATE
);

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 DISTINCT B.BOOK_ID, B.NAME
FROM BOOKS B
WHERE AVAILABLE_FROM < '2019-05-23' AND BOOK_ID NOT IN
(SELECT BOOK_ID FROM ORDERS2
WHERE DISPATCH_DATE BETWEEN '2018-06-23' AND '2019-06-23'
GROUP BY BOOK_ID HAVING SUM(QUANTITY) >=10
);
```

Q49. Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course_id.
Return the result table ordered by student_id in ascending order.

Solution:

```
CREATE TABLE ENROLLMENTS
(
STUDENT_ID INT,
COURSE_ID INT,
GRADE INT,
PRIMARY KEY (STUDENT_ID,COURSE_ID)
);

INSERT INTO ENROLLMENTS VALUES
(2, 2, 95), (2, 3, 95), (1, 1, 90), (1, 2, 99), (3, 1, 80), (3, 2, 75), (3, 3, 82);

SELECT E1.STUDENT_ID, E1.COURSE_ID AS COURSE, E1.GRADE
FROM ENROLLMENTS E1
WHERE GRADE = (SELECT MAX(GRADE) AS MAX_GRADE
FROM ENROLLMENTS E2
WHERE E1.STUDENT_ID = E2.STUDENT_ID)
GROUP BY E1.STUDENT_ID, E1.GRADE
ORDER BY E1.STUDENT_ID ASC;

SELECT E1.STUDENT_ID, E1.COURSE_ID AS COURSE, E1.GRADE
FROM (
  SELECT
    E.STUDENT_ID, E.COURSE_ID, E.GRADE,
    RANK() OVER (PARTITION BY E.STUDENT_ID ORDER BY E.GRADE DESC) AS RNK
  FROM ENROLLMENTS E
) E1
WHERE E1.RNK = 1
GROUP BY STUDENT_ID
ORDER BY E1.STUDENT_ID ASC;
```

Q50. Write an SQL query to find the winner in each group. Return the result table in any order.

Solution:

```
CREATE TABLE TEAMS
(
TEAM_ID INT PRIMARY KEY,
GROUP_ID INT
);

INSERT INTO TEAMS VALUES (15,1),(25,1),(30,1),(45,1),(10,2),(35,2),(50,2),(20,3),(40,3);

CREATE TABLE MATCHES
(
MATCH_ID INT PRIMARY KEY,
FIRST_PLAYER INT,
SECOND_PLAYER INT,
FIRST_SCORE INT,
SECOND_SCORE INT
);

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 T.GROUP_ID, MAX(P.TEAM_ID) AS WINNER
FROM (
  SELECT M.FIRST_PLAYER AS TEAM_ID, SUM(CASE
                                  WHEN M.FIRST_SCORE > M.SECOND_SCORE THEN 3
                                  WHEN M.FIRST_SCORE = M.SECOND_SCORE THEN 1
                                  ELSE 0
                                 END) AS TOTAL_POINTS
  FROM MATCHES M
  GROUP BY M.FIRST_PLAYER
  UNION ALL
  SELECT M.SECOND_PLAYER AS TEAM_ID, SUM(CASE
                                   WHEN M.SECOND_SCORE > M.FIRST_SCORE THEN 3
```

```
                        WHEN M.FIRST_SCORE = M.SECOND_SCORE THEN 1
                        ELSE 0
                       END) AS TOTAL_POINTS
  FROM MATCHES M
  GROUP BY M.SECOND_PLAYER
) P
JOIN TEAMS T ON T.TEAM_ID = P.TEAM_ID
GROUP BY T.GROUP_ID ;
```