**Question 1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA.**

|  |
| --- |
| SELECT \*  FROM city  WHERE population > 100000  AND countryCode = 'USA'; |

**Question 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'; |

**Question 3. Query all columns (attributes) for every row in the CITY table.**

|  |
| --- |
| SELECT \* FROM city; |

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

|  |
| --- |
| SELECT \*  FROM city  WHERE id = 1661; |

**Question 5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.**

|  |
| --- |
| SELECT \*  FROM city  WHERE countryCode = 'JPN'; |

**Question 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'; |

**Question 7. Query a list of CITY and STATE from the STATION table**

|  |
| --- |
| SELECT city, state  FROM station; |

**Question 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 MOD(id, 2)=0  ORDER BY city; |

**Question 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 diff\_between\_city\_and\_distinct\_city  FROM station; |

**Question 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.**

|  |
| --- |
| ( SELECT city, LENGTH(city) AS smallestCityName  FROM station  ORDER BY LENGTH(city) ASC, city LIMIT 1 )  UNION  ( SELECT city, LENGTH(city) AS largestCityName  FROM station  ORDER BY LENGTH(city) DESC, city LIMIT 1 ); |

**Question 11. 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  FROM station  WHERE (city LIKE 'a%')  OR (city LIKE 'e%')  OR (city LIKE 'i%')  OR (city LIKE 'o%')  OR (city LIKE 'u%'); |

**Question 12. 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 (city LIKE '%a')  OR (city LIKE '%e')  OR (city LIKE '%i')  OR (city LIKE '%o')  OR (city LIKE '%u'); |

**Question 13. 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 (city NOT LIKE 'a%')  AND (city NOT LIKE 'e%')  AND (city NOT LIKE 'i%')  AND (city NOT LIKE 'o%')  AND (city NOT LIKE 'u%'); |

**Question 14. 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 (city NOT LIKE '%a')  AND (city NOT LIKE '%e')  AND (city NOT LIKE '%i')  AND (city NOT LIKE '%o')  AND (city NOT LIKE '%u'); |

**Question 15. 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 LIKE 'a%' AND city NOT LIKE 'e%' AND city NOT LIKE 'i%' AND city NOT LIKE 'o%' AND city NOT LIKE 'u%')  AND (city NOT LIKE '%a' AND city NOT LIKE '%e' AND city NOT LIKE '%i' AND city NOT LIKE '%o' AND city NOT LIKE '%u'); |

**Question 16. 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 LIKE 'a%' AND city NOT LIKE 'e%' AND city NOT LIKE 'i%' AND city NOT LIKE 'o%' AND city NOT LIKE 'u%')  AND (city NOT LIKE '%a' AND city NOT LIKE '%e' AND city NOT LIKE '%i' AND city NOT LIKE '%o' AND city NOT LIKE '%u'); |

**Question 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.**

|  |
| --- |
| 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  HAVING SUM(IF(s.sale\_date BETWEEN '2019-01-01' AND '2019-03-31', 1, 0)) = SUM(IF(s.sale\_date, 1, 0)); |

**Question 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.**

|  |
| --- |
| SELECT DISTINCT v2.viewer\_id AS id  FROM views v1  INNER JOIN views v2  ON v1.author\_id = v2.viewer\_id  ORDER BY id ASC;  **OR**  SELECT author\_id AS id  FROM views  WHERE author\_id = viewer\_id  GROUP BY author\_id  ORDER BY author\_id ASC; |

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

|  |
| --- |
| SELECT ROUND((d2.immediate\_orders/COUNT(d1.delivery\_id))\*100, 2) AS immediate\_percentage  FROM delivery d1,  (SELECT COUNT(order\_date) AS immediate\_orders  FROM delivery  WHERE order\_date = customer\_pref\_delivery\_date) d2; |

**Question 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.**

|  |
| --- |
| SELECT ad\_id,  IFNULL(ROUND((SUM(action ='Clicked')/SUM(action !='ignored'))\*100, 2), 0) AS ctr  FROM ads  GROUP BY ad\_id  ORDER BY ctr DESC, ad\_id ASC; |

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

|  |
| --- |
| SELECT e.employee\_id,  (SELECT COUNT(team\_id) FROM employee  WHERE team\_id = e.team\_id) AS team\_size  FROM employee e; |

**Question 22. Write an SQL query to find the type of weather in each country for November 2019. The type of weather is:**

* **Cold if the average weather\_state is less than or equal 15,**
* **Hot if the average weather\_state is greater than or equal to 25, and**
* **Warm otherwise.**

**Return result table in any order**

|  |
| --- |
| SELECT c.country\_name,  CASE WHEN AVG(w.weather\_state) <= 15 THEN "Cold"  WHEN AVG(w.weather\_state) >= 25 THEN "Hot"  ELSE "Warm"  END AS weather\_type  FROM countries c  INNER JOIN weather w  ON c.country\_id = w.country\_id  WHERE LEFT(day, 7) = '2019-11'  GROUP BY country\_name; |

**Question 23. Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places.**

|  |
| --- |
| SELECT p.product\_id,  ROUND(CAST(SUM(p.price\*us.units) AS DECIMAL)/SUM(us.units),2) AS average\_price  FROM prices AS p, UnitsSold AS us  WHERE us.purchase\_date BETWEEN p.start\_date AND p.end\_date  AND p.product\_id = us.product\_id  GROUP BY p.product\_id; |

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

|  |
| --- |
| SELECT player\_id,  MIN(event\_date) AS first\_login  FROM Activity  GROUP BY player\_id; |

**Question 25. Write an SQL query to report the device that is first logged in for each player. Return the result table in any order.**

|  |
| --- |
| SELECT player\_id,  device\_id  FROM Activity  WHERE (player\_id, event\_date) IN  (select player\_id, MIN(event\_date)  FROM Activity  GROUP BY player\_id); |

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

|  |
| --- |
| SELECT p.product\_name,  SUM(o.unit) AS unit  FROM products p  INNER JOIN orders o  ON p.product\_id = o.product\_id  WHERE o.order\_date BETWEEN '2020-02-01' AND '2020-02-29'  GROUP BY p.product\_id  HAVING unit >= 100; |

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

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

|  |
| --- |
| SELECT \*  FROM users  WHERE REGEXP\_LIKE(mail, '^[a-zA-Z][a-zA-Z0-9\\_\.\-]\*@leetcode.com'); |

**Question 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.**

|  |
| --- |
| 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  ); |

**Question 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.**

|  |
| --- |
| SELECT DISTINCT c.title  FROM content c  JOIN TVProgram t  ON c.content\_id = t.content\_id  WHERE c.kids\_content = 'Y'  AND c.content\_type = 'Movies'  AND MONTH(t.program\_date) = 6  AND YEAR(t.program\_date) = 2020; |

**Question 30. Write an SQL query to find the npv of each query of the Queries table.**

|  |
| --- |
| SELECT q.id,  q.year,  IFNULL(n.npv, 0) AS npv  FROM queries q  LEFT JOIN npv n  ON q.id = n.id AND q.year = n.year; |

**Question 31. Write an SQL query to find the npv of each query of the Queries table.**

Duplicate

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

|  |
| --- |
| SELECT IFNULL(unique\_id, 'null') as unique\_id,  name  FROM employees  LEFT JOIN employeeUNI  ON employees.id = employeeUNI.id; |

**Question 33. Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled\_distance in descending order, if two or more users travelled the same distance, order them by their name in ascending order.**

|  |
| --- |
| SELECT u.name,  SUM(IFNULL(r.distance, 0)) AS travelled\_distance  FROM rides r  RIGHT JOIN users u  ON r.user\_id = u.id  GROUP BY name  ORDER BY travelled\_distance desc, u.name asc; |

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

Duplicate

**Question 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.**

|  |
| --- |
| ( SELECT name  FROM users u  JOIN  ( SELECT user\_id,  COUNT(1) AS count  FROM movie\_rating  GROUP BY user\_id  ORDER BY count  LIMIT 1 ) tmp1  ON u.user\_id = tmp1.user\_id )  UNION  ( SELECT title  FROM movies m  JOIN  ( SELECT movie\_id,  AVG(rating) AS avg\_rating  FROM movie\_rating  WHERE created\_at BETWEEN '2020-02-01' AND '2020-02-29'  GROUP BY movie\_id  ORDER BY avg\_rating  LIMIT 1 ) tmp2  ON m.movie\_id = tmp2.movie\_id ); |

**Question 36. Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled\_distance in descending order, if two or more users travelled the same distance, order them by their name in ascending order.**

Duplicate

**Question 37. 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.**

Duplicate

**Question 38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.**

|  |
| --- |
| SELECT id,  name  FROM students  WHERE department\_id NOT IN  ( SELECT id  FROM departments ); |

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

|  |
| --- |
| SELECT from\_id AS person1,  to\_id AS person2,  COUNT(duration) AS call\_count,  SUM(duration) AS total\_duration  FROM ( SELECT \*  FROM calls  UNION ALL  SELECT to\_id,  from\_id,  duration  FROM calls ) t1  WHERE person1 < person2  GROUP BY person1, person2; |

**Question 40. Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places.**

Duplicate 23

**Question 41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse.**

|  |
| --- |
| SELECT name AS warehouse\_name,  SUM(units\*width\*length\*height) AS volume  FROM warehouse w  INNER JOIN products p  ON w.product\_id = p.product\_id  GROUP BY name  ORDER BY name; |

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

|  |
| --- |
| SELECT s1.sale\_date,  s1.sold\_num - s2.sold\_num AS diff  FROM sales s1  LEFT JOIN sales s2  ON s1.sale\_date = s2.sale\_date  WHERE s1.fruit = 'apples'  AND s2.fruit = 'oranges'; |

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

|  |
| --- |
| WITH cte AS  (  SELECT a1.player\_id AS player\_id  FROM activity a1  RIGHT JOIN activity a2  ON DATEADD(DAY, 1, a1.event\_date) = a2.event\_date  )  SELECT ROUND(COUNT(DISTINCT cte.player\_id)/COUNT(DISTINCT activity.player\_id), 2) AS fraction  FROM activity, cte; |

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

|  |
| --- |
| SELECT name  FROM employee e  JOIN  ( SELECT managerId  FROM employee  GROUP BY managerId  HAVING COUNT(managerId) >= 5 ) tmp  ON e.id = tmp.managerId; |

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

|  |
| --- |
| SELECT d.dept\_name,  COUNT(1) AS student\_number  FROM department d  LEFT OUTER JOIN student s  ON d.dept\_id = s.dept\_id  GROUP BY d.dept\_name  ORDER BY student\_number DESC, d.dept\_name; |

**Question 46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.**

|  |
| --- |
| SELECT customer\_id  FROM customer  GROUP BY customer\_id  HAVING COUNT(DISTINCT product\_key) = (SELECT COUNT(1) FROM product); |

**Question 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.**

|  |
| --- |
| 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 p  JOIN employee e  ON p.employee\_id = e.employee\_id ) tmp  WHERE rnk = 1; |

**Question 48. Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23.**

|  |
| --- |
| SELECT book\_id,  name  FROM books  WHERE available\_from < '2019-05-23'  AND book\_id NOT IN  ( SELECT book\_id  FROM orders  WHERE dispatch\_date BETWEEN '2018-06-23' AND '2019-06-23'  GROUP BY book\_id  HAVING SUM(quantity) >= 10 ); |

**Question 49. Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course\_id. Return the result table ordered by student\_id in ascending order.**

|  |
| --- |
| SELECT student\_id,  MIN(course\_id) AS course\_id,  grade  FROM enrollments  WHERE (student\_id, grade) IN  ( SELECT student\_id,  MAX(grade)  FROM enrollments  GROUP BY student\_id )  GROUP BY student\_id  ORDER BY student\_id ASC; |

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

|  |
| --- |
| SELECT group\_id,  player\_id  FROM  ( SELECT group\_id,  player\_id,  SUM (  ( CASE WHEN 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 ) temp  GROUP BY group\_id  ORDER BY group\_id, totalScores DESC, player\_id; |