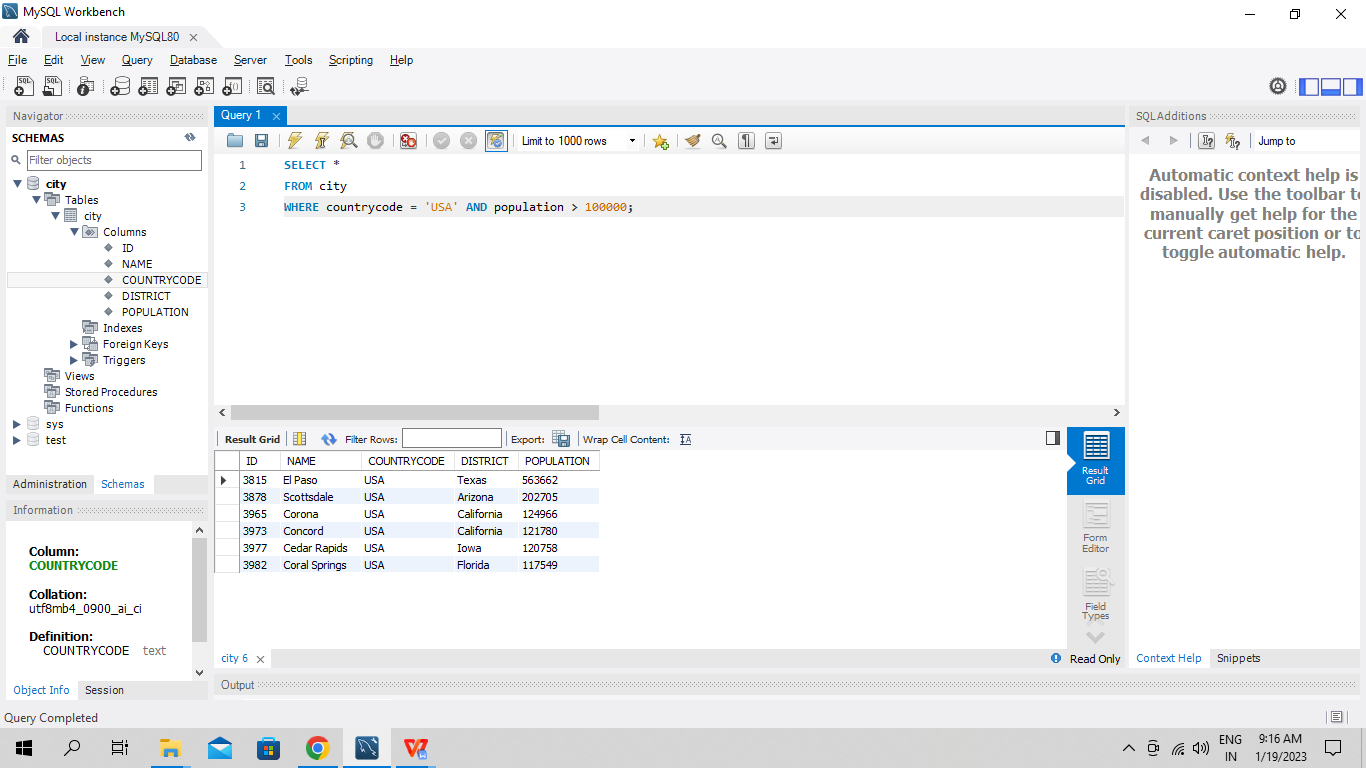
**Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA.**

SELECT \*

FROM city

WHERE countrycode = 'USA' AND population > 100000;

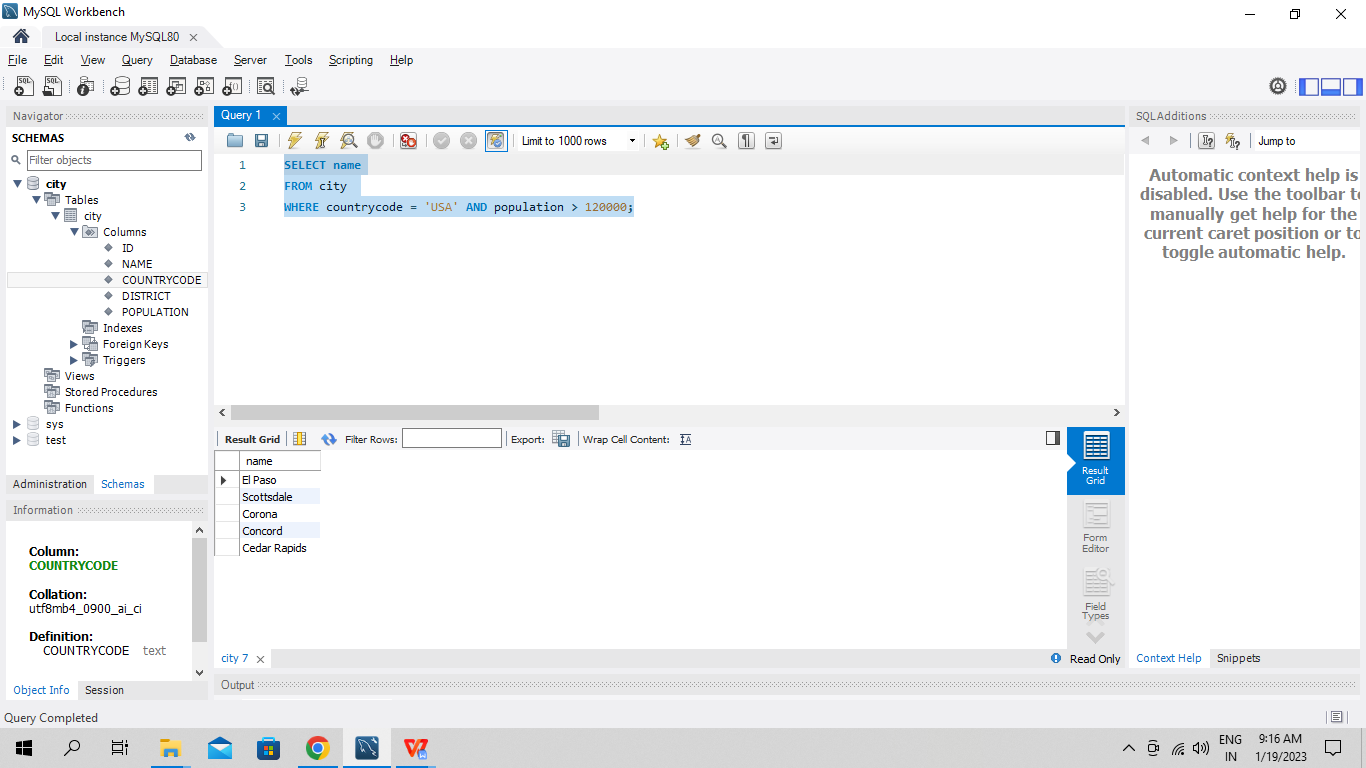


**Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.**

SELECT name

FROM city

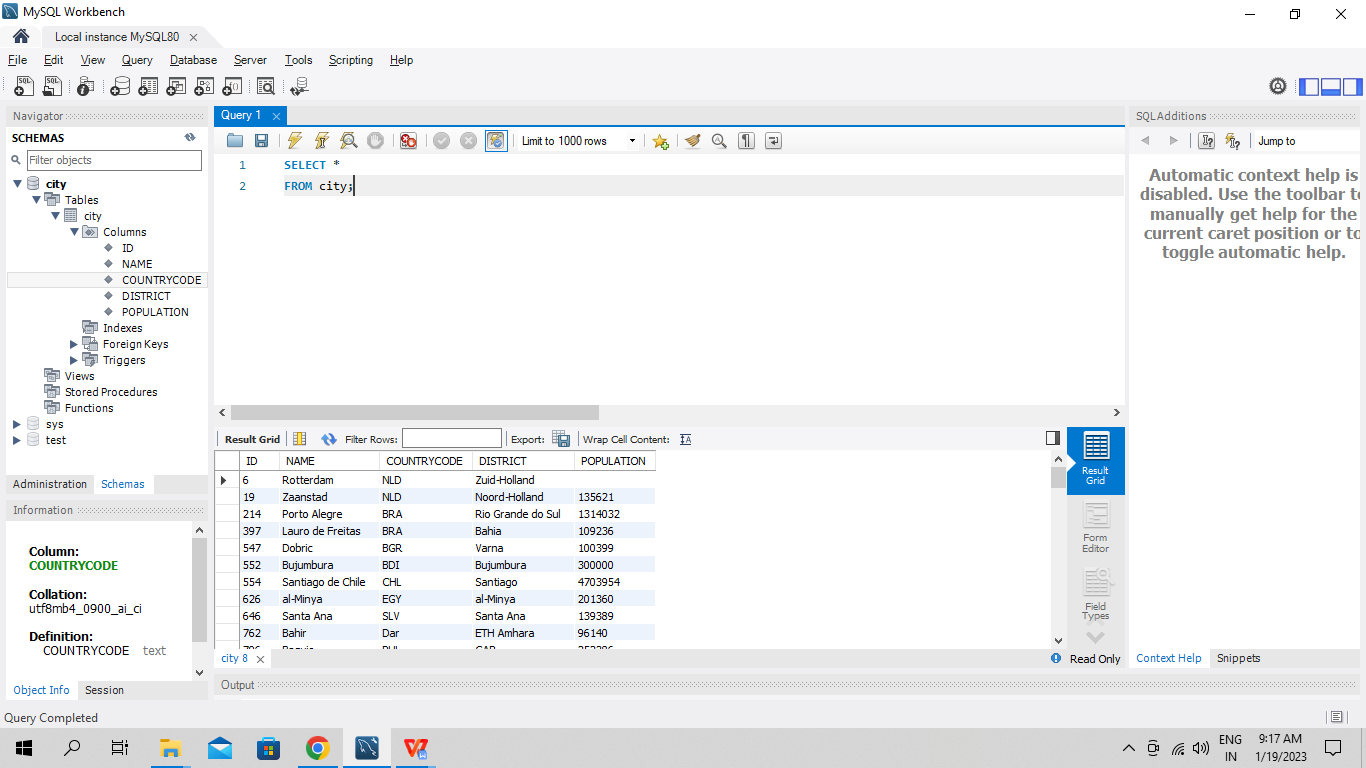
WHERE countrycode = 'USA' AND population > 120000;



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

SELECT \*

FROM city;

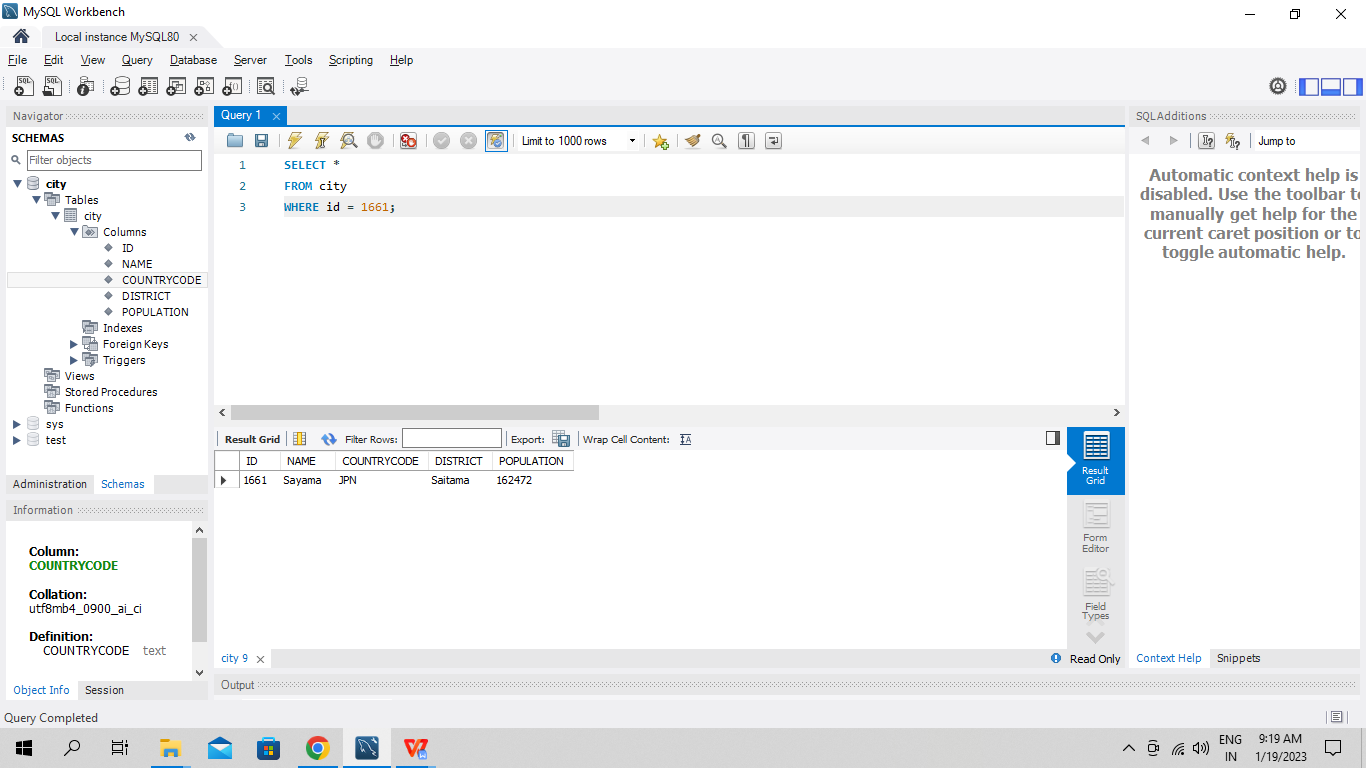


**Q4. Query all columns for a city in CITY with the ID 1661.**

SELECT \*

FROM city

WHERE id = 1661;

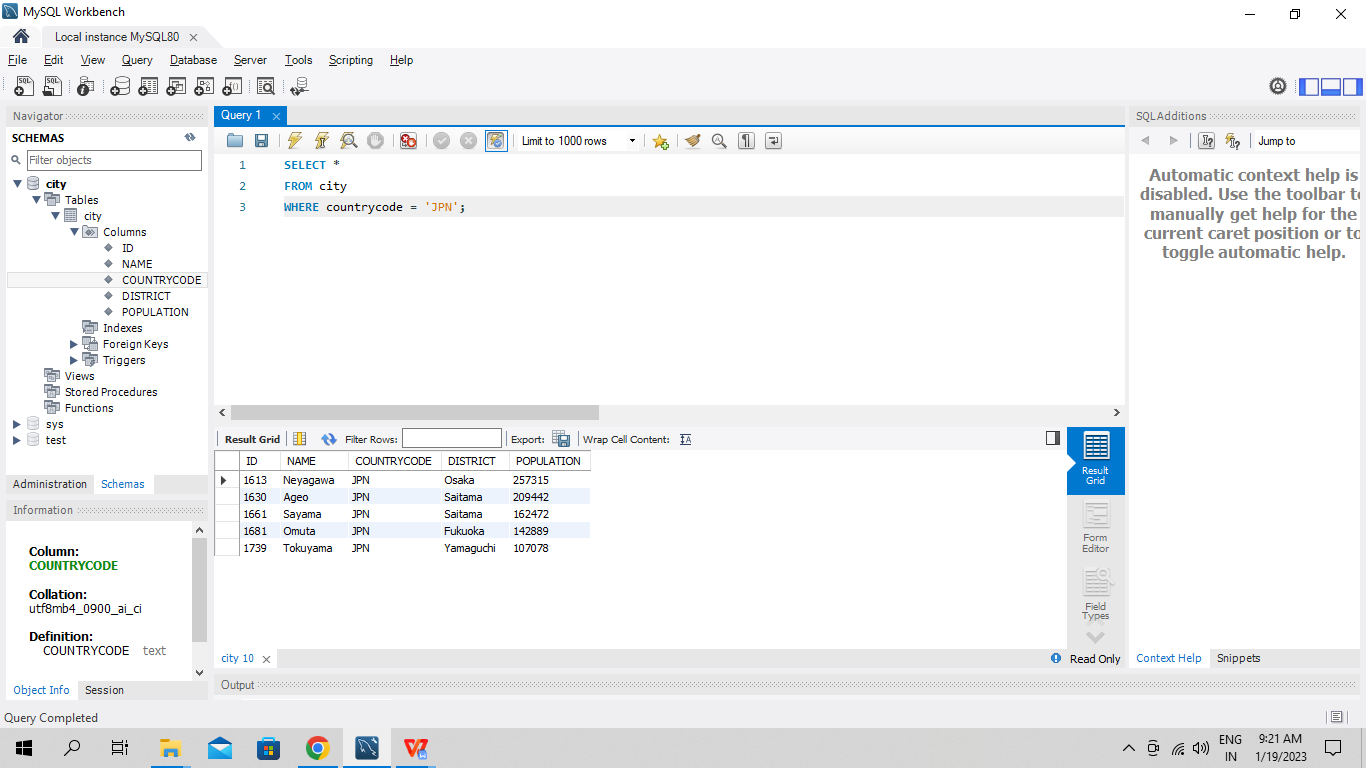


**Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.**

SELECT \*

FROM city

WHERE countrycode = 'JPN';

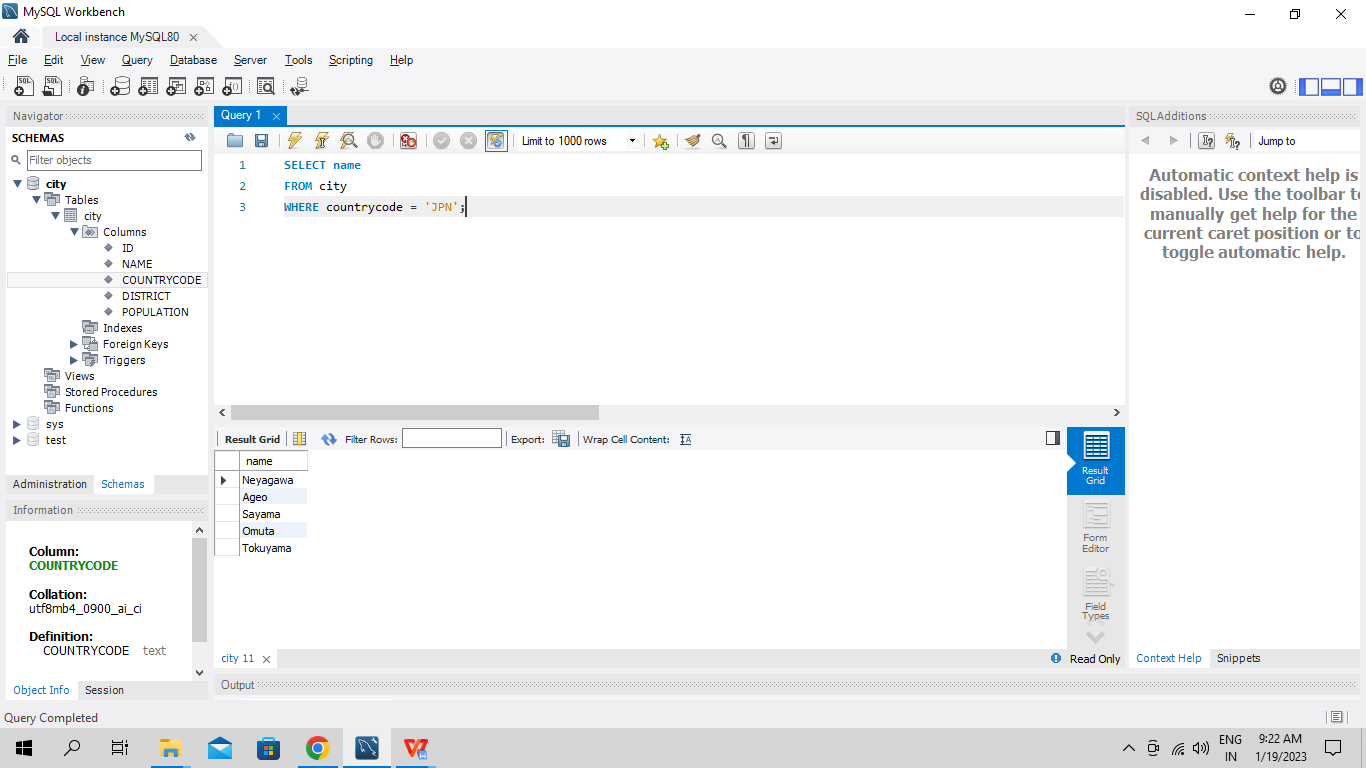


**Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is JPN.**

SELECT name

FROM city

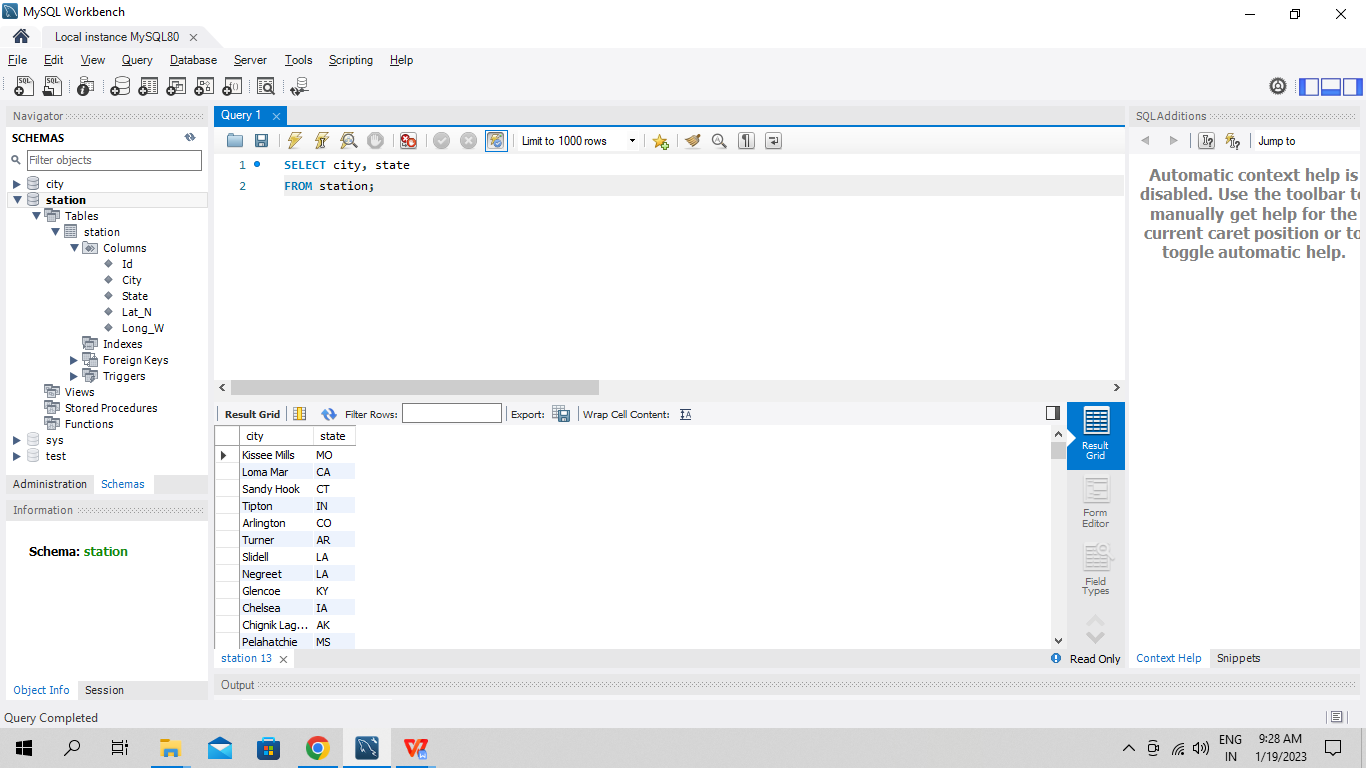
WHERE countrycode = 'JPN';



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

**SELECT city, state**

**FROM station;**

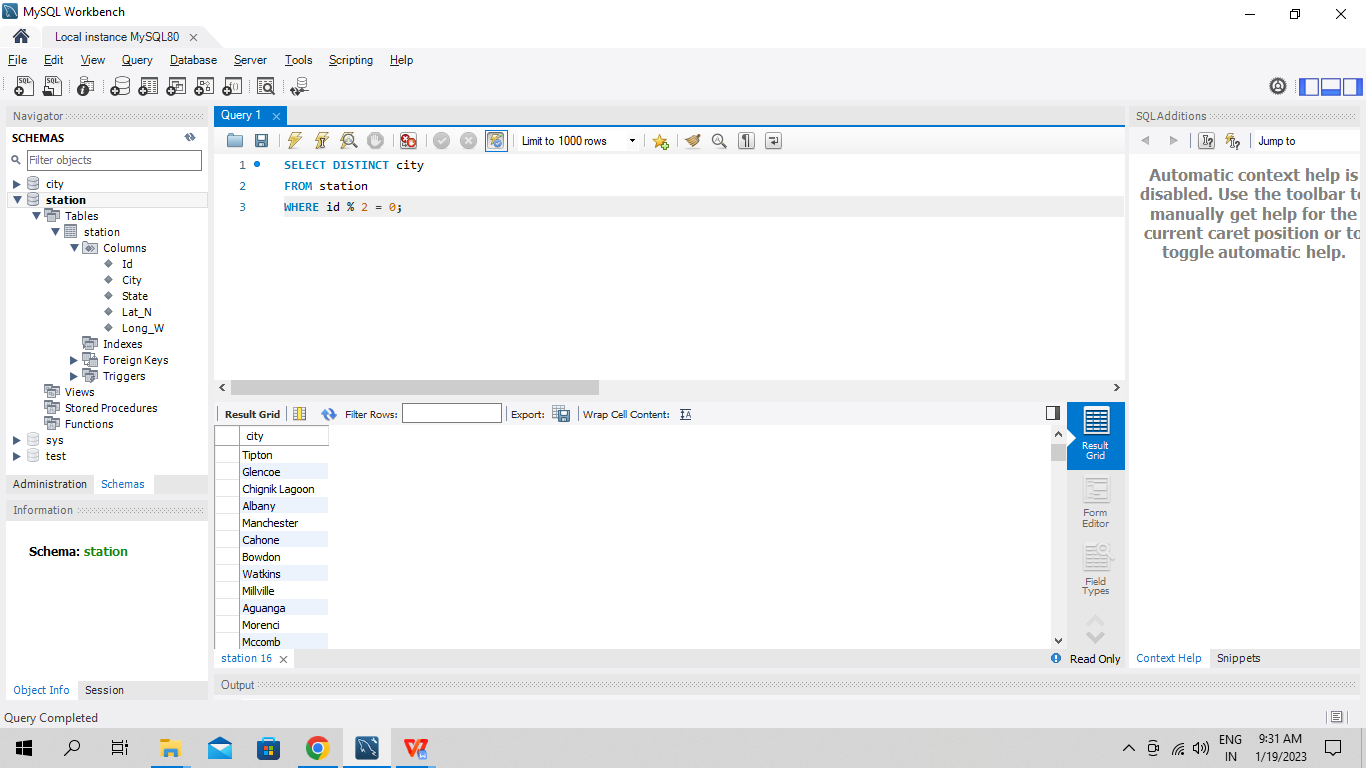


**Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order, but exclude duplicates from the answer.**

SELECT DISTINCT city

FROM station

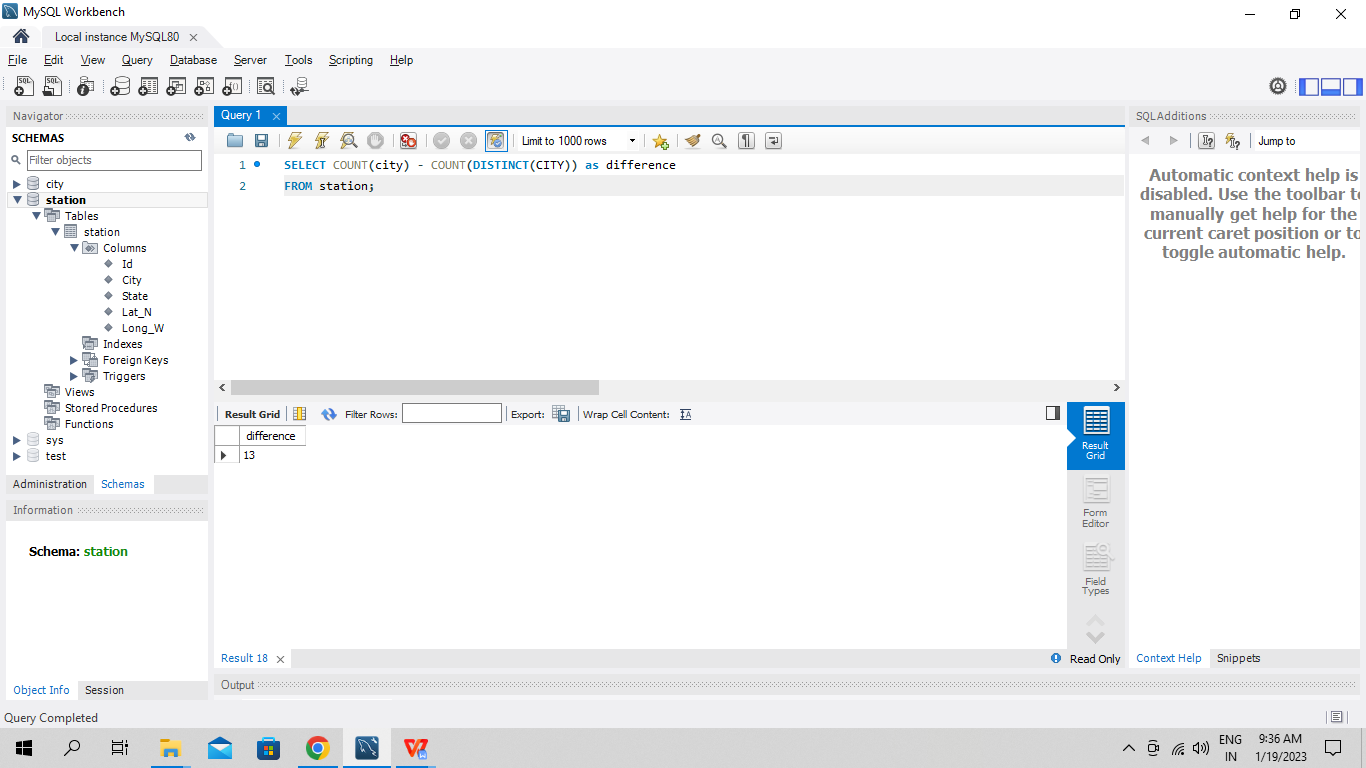
WHERE id % 2 = 0;



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

SELECT COUNT(city) - COUNT(DISTINCT(CITY)) as difference

FROM station;



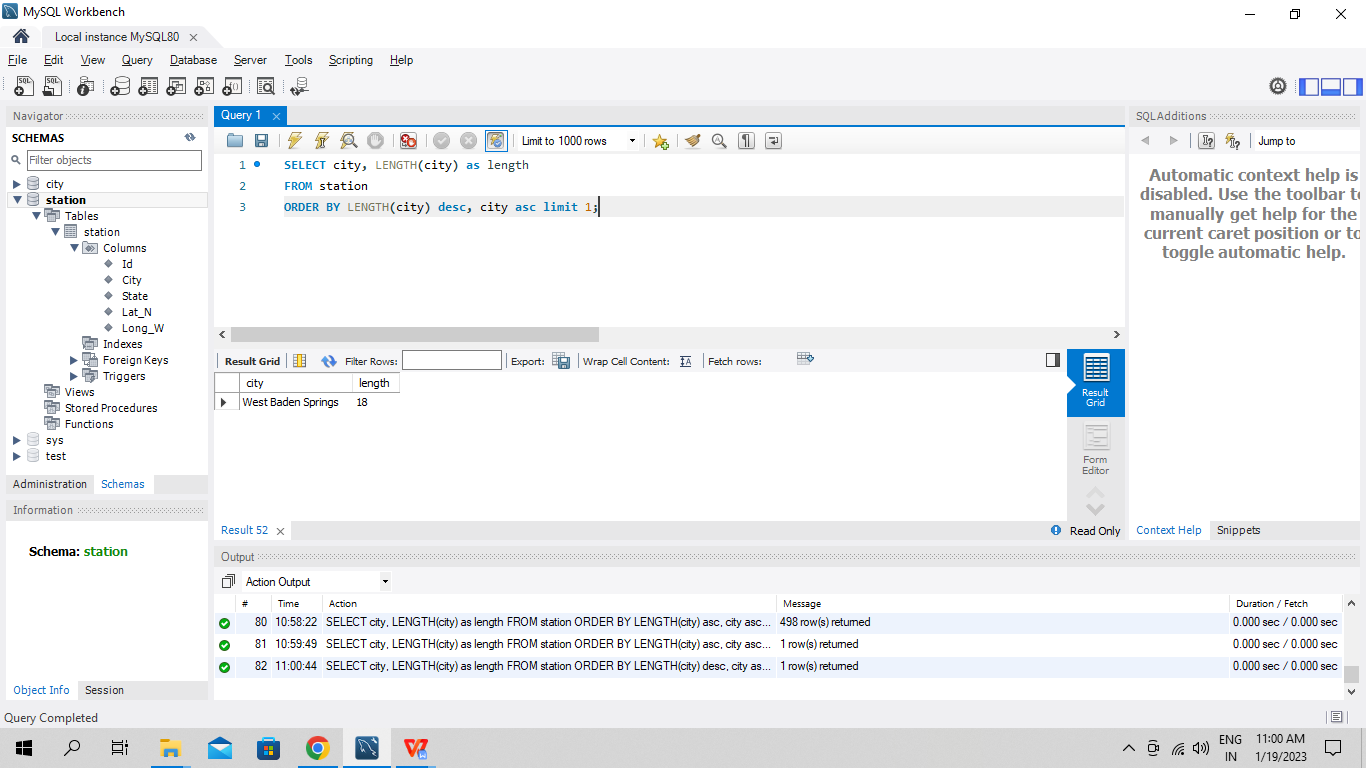
**Q10. Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.**

1. **Longest CITY name -**

SELECT city, LENGTH(city) as length

FROM station

ORDER BY LENGTH(city) desc, city asc limit 1;

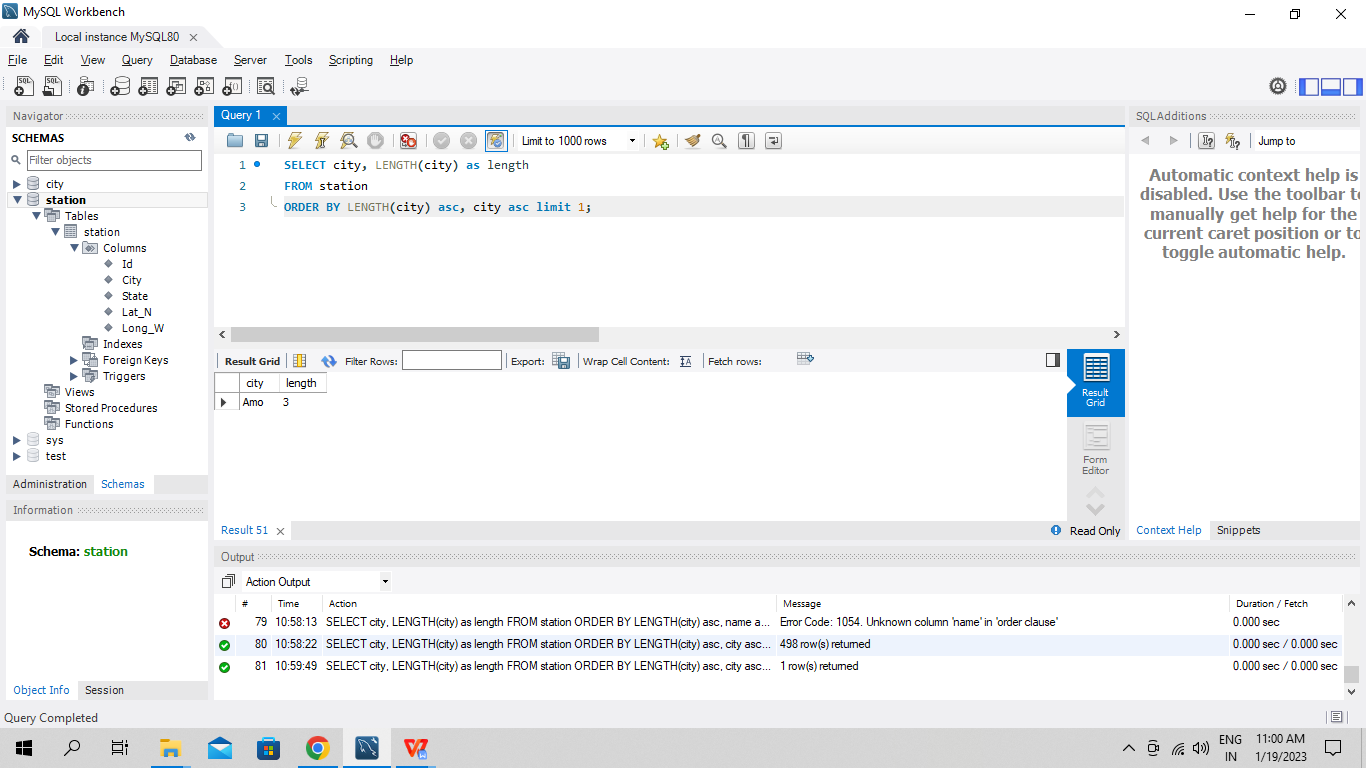


1. **Shortest CITY name -**

SELECT city, LENGTH(city) as length

FROM station

ORDER BY LENGTH(city) asc, city asc 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)

FROM station

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

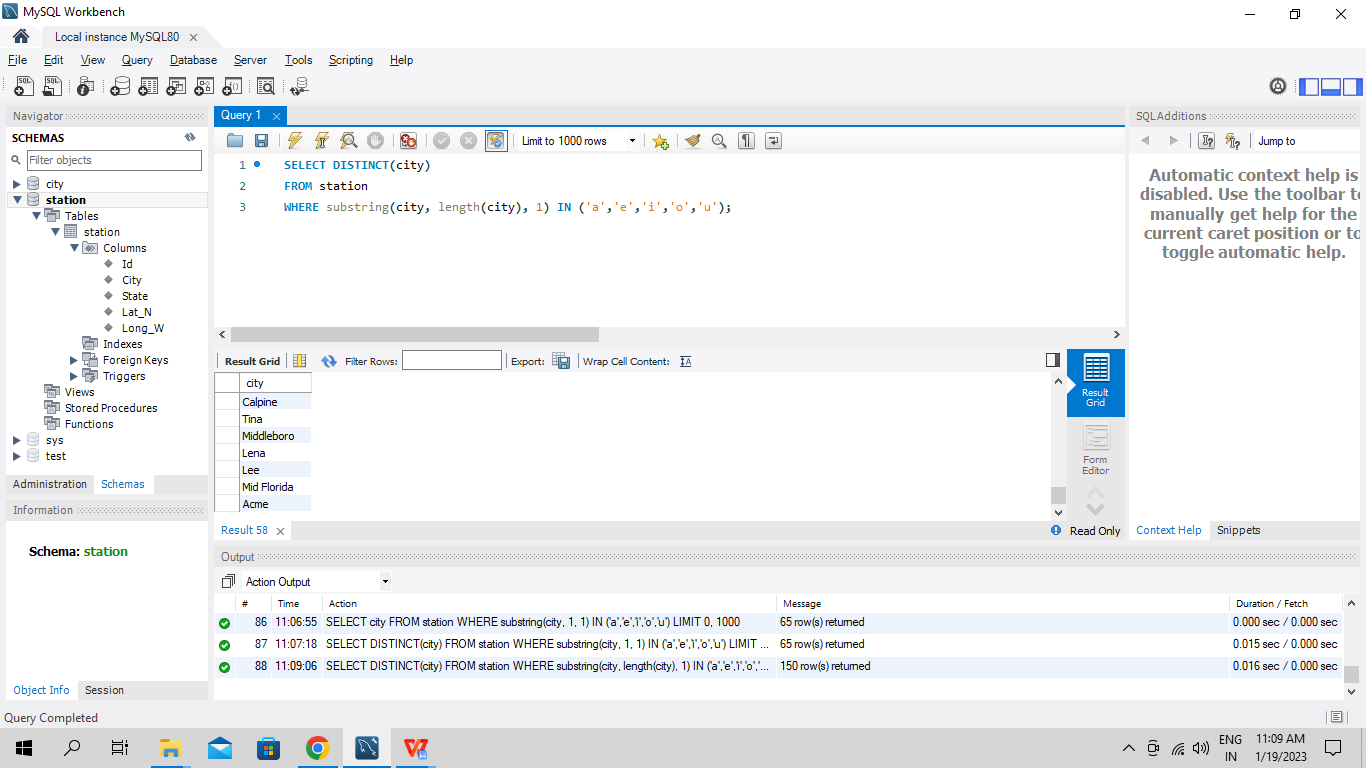


**Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.**

SELECT DISTINCT(city)

FROM station

WHERE substring(city, length(city), 1) IN ('a','e','i','o','u');

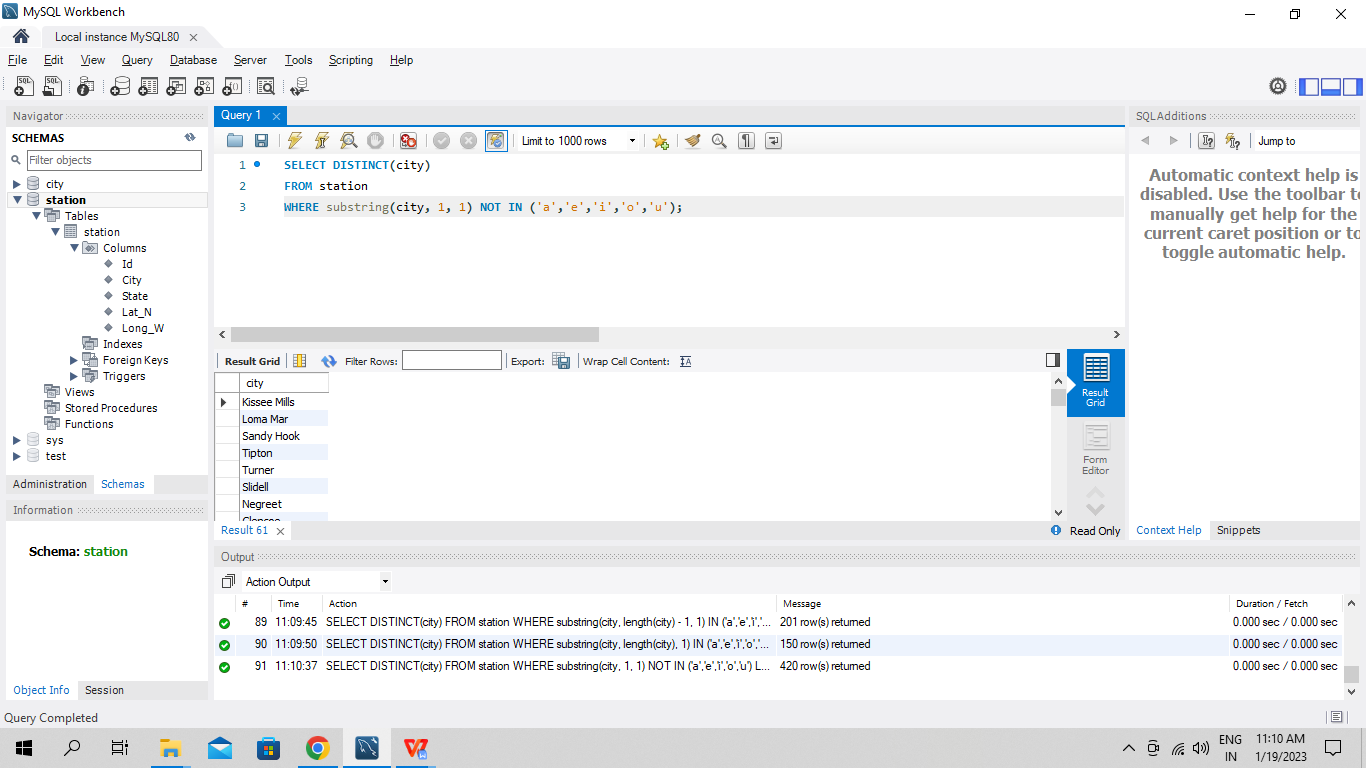


**Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.**

SELECT DISTINCT(city)

FROM station

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

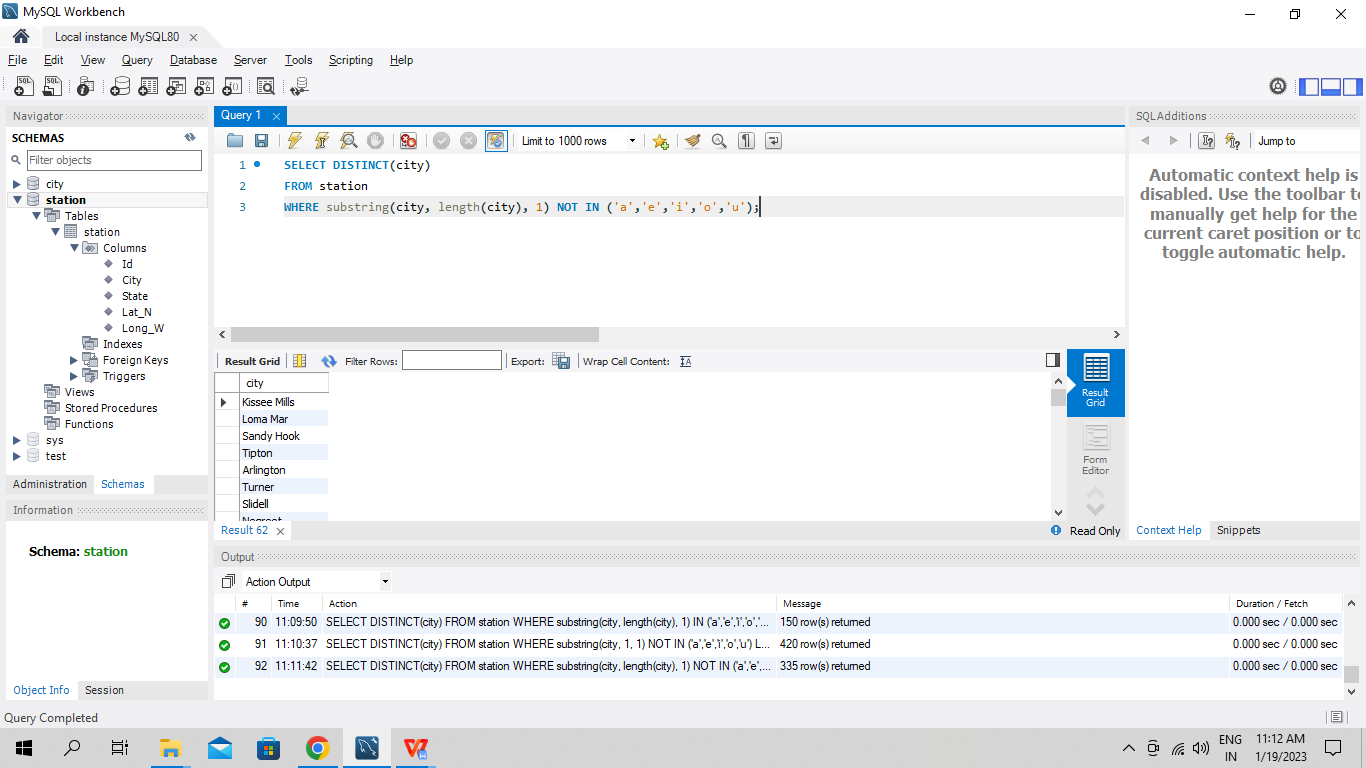


**Q14. Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.**

SELECT DISTINCT(city)

FROM station

WHERE substring(city, length(city), 1) NOT IN ('a','e','i','o','u');

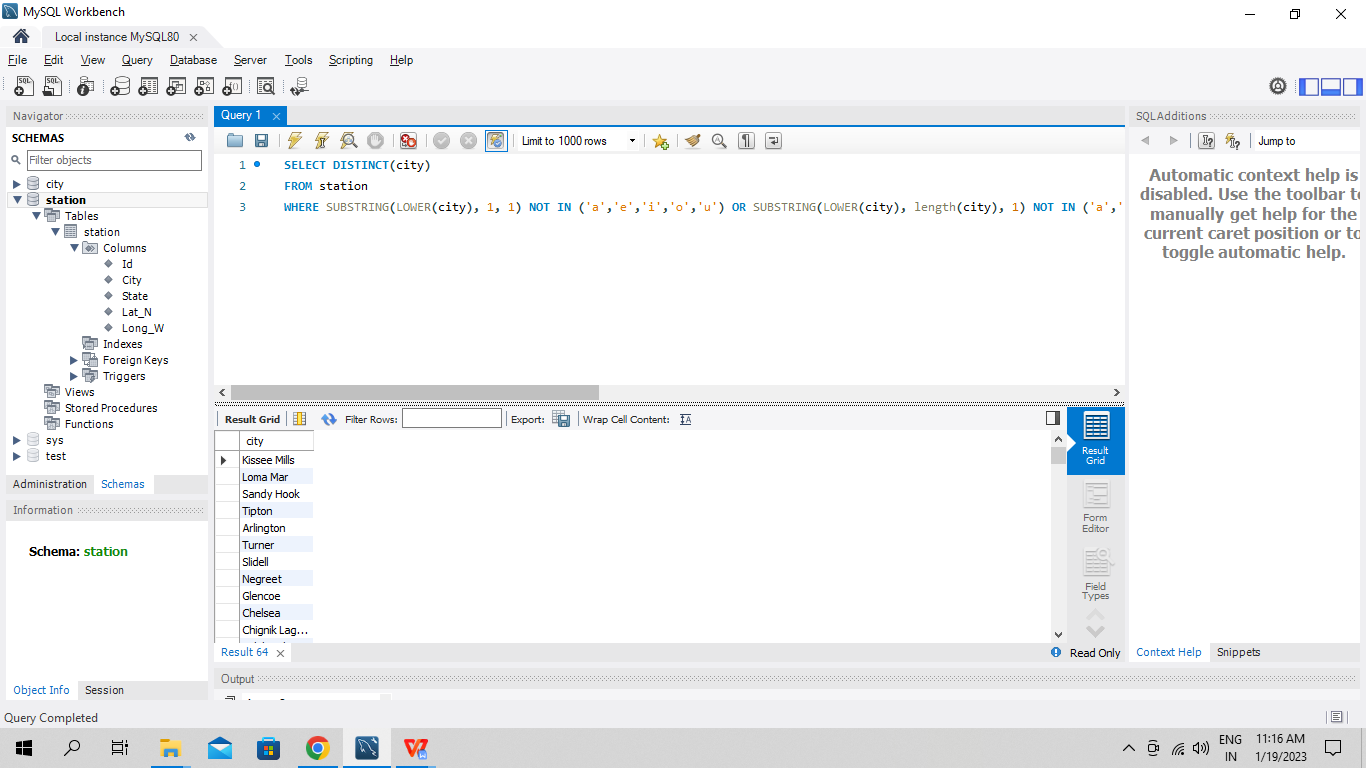


**Q15. Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.**

SELECT DISTINCT(city)

FROM station

WHERE SUBSTRING(LOWER(city), 1, 1) NOT IN ('a','e','i','o','u') OR SUBSTRING(LOWER(city), length(city), 1) NOT IN ('a','e','i','o','u');



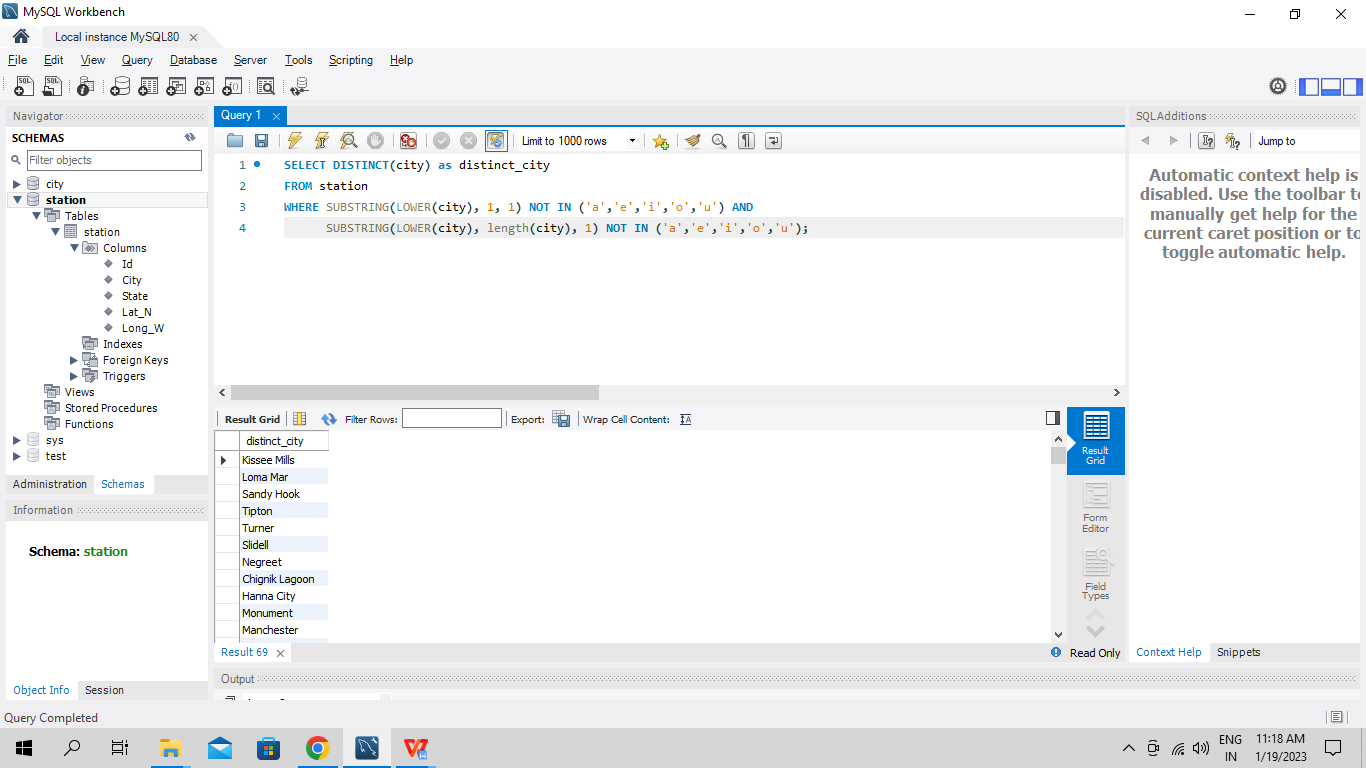
**Q16. Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.**

SELECT DISTINCT(city) as distinct\_city

FROM station

WHERE SUBSTRING(LOWER(city), 1, 1) NOT IN ('a','e','i','o','u') AND

SUBSTRING(LOWER(city), length(city), 1) NOT IN ('a','e','i','o','u');

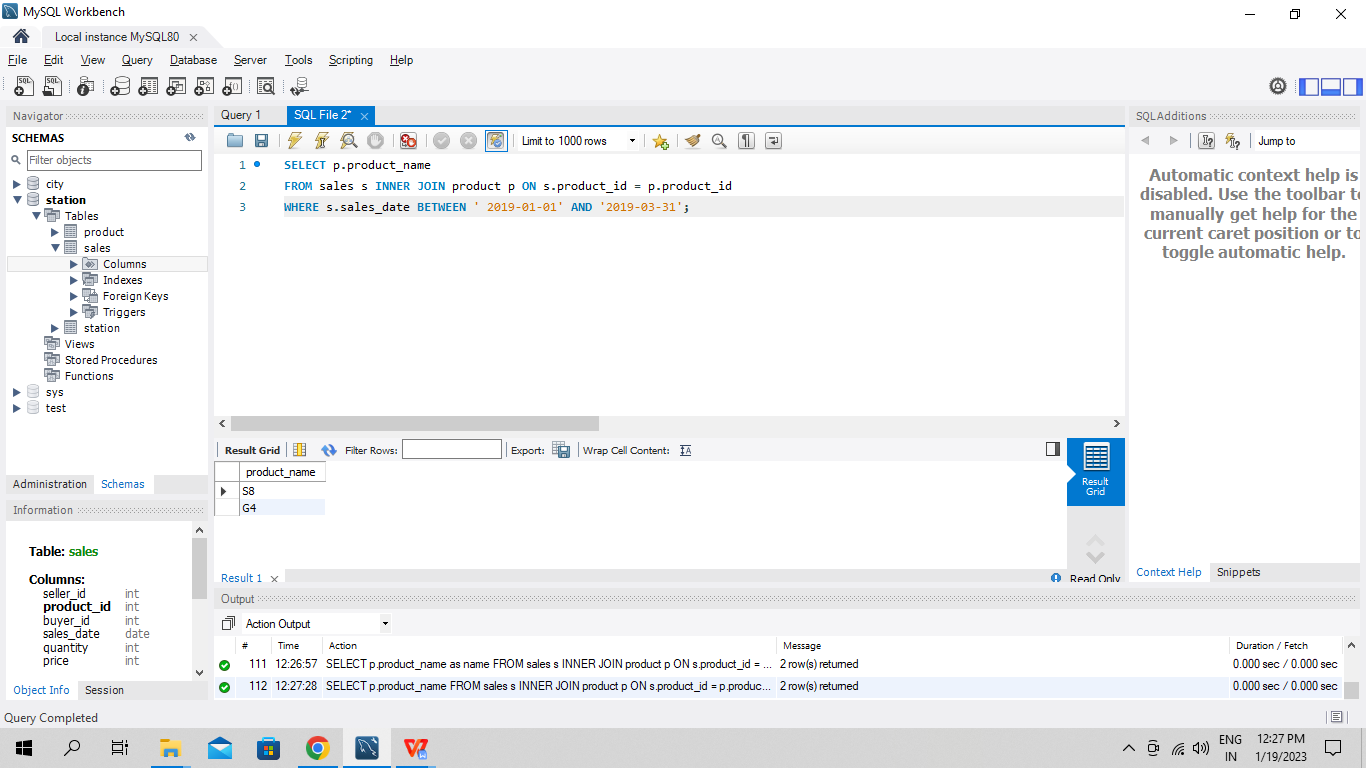


**Q17 Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive. Return the result table in any order.**

SELECT p.product\_name

FROM sales s INNER JOIN product p ON s.product\_id = p.product\_id

WHERE s.sales\_date BETWEEN ' 2019-01-01' AND '2019-03-31';

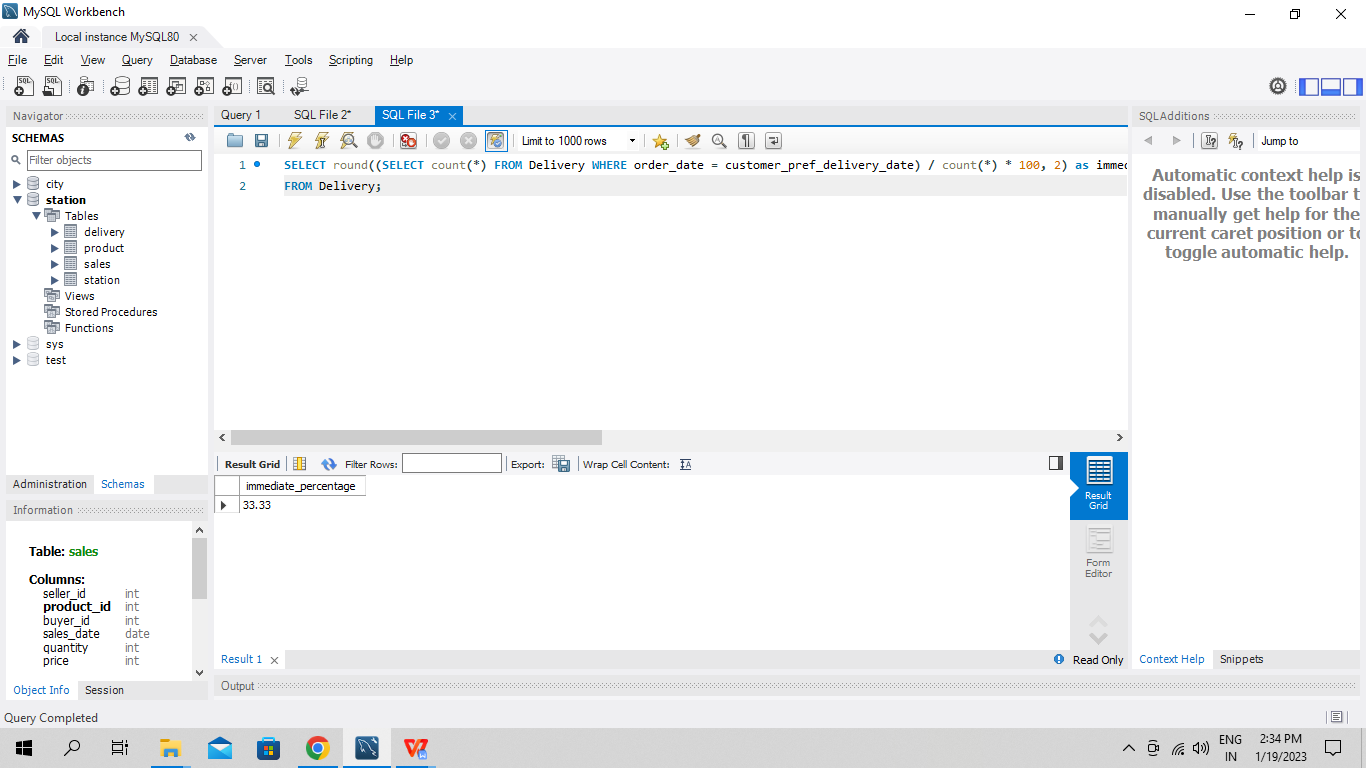


**Q18. Write an SQL query to find all the authors that viewed at least one of their own articles. Return the result table sorted by id in ascending order.**

**Q19. 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.**

SELECT round((SELECT count(\*) FROM Delivery WHERE order\_date = customer\_pref\_delivery\_date) / count(\*) \* 100, 2) as immediate\_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.**

SELECT ad\_id,

ROUND (IFNULL ( AVG ( CASE

WHEN action = 'Clicked' THEN 1

WHEN action = 'Viewed' THEN 0

ELSE NULL

END

) \* 100

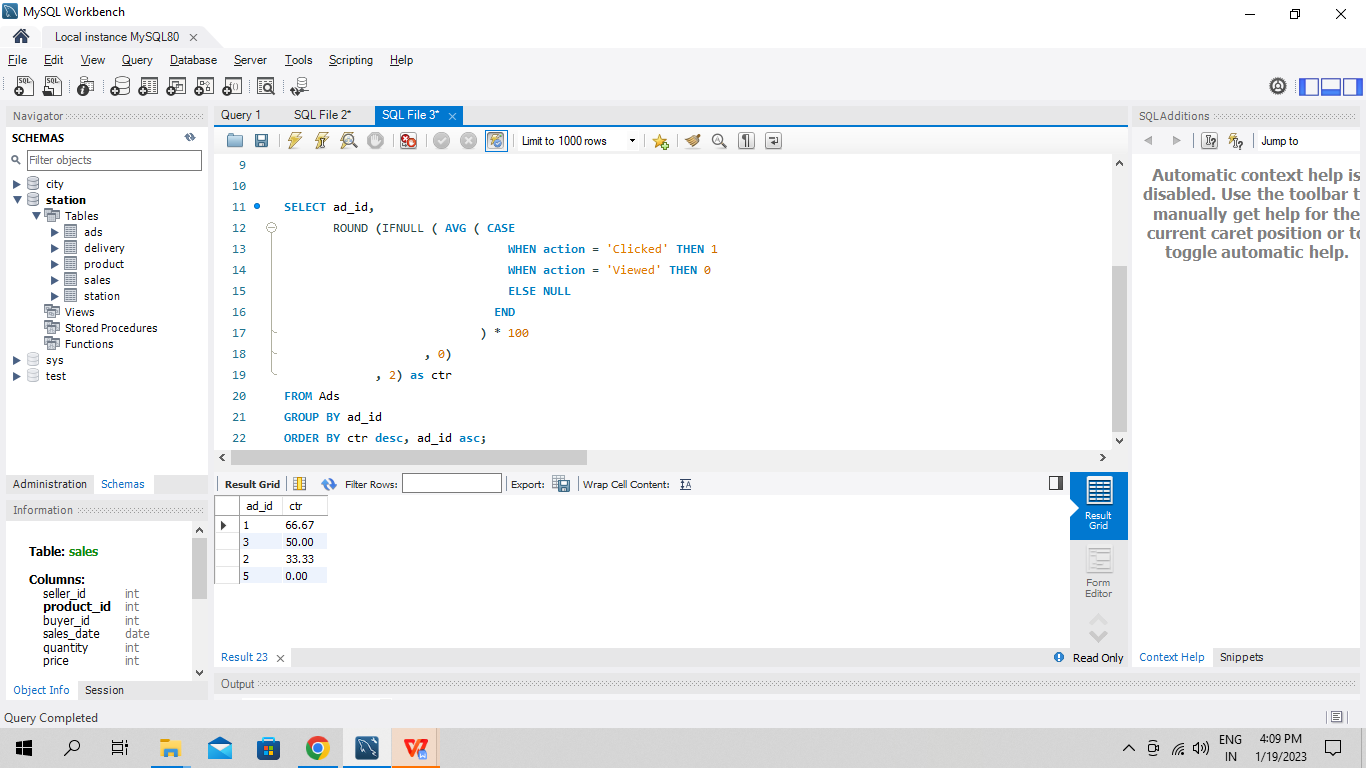
, 0)

, 2) as ctr

FROM Ads

GROUP BY ad\_id

ORDER BY ctr desc, ad\_id asc;

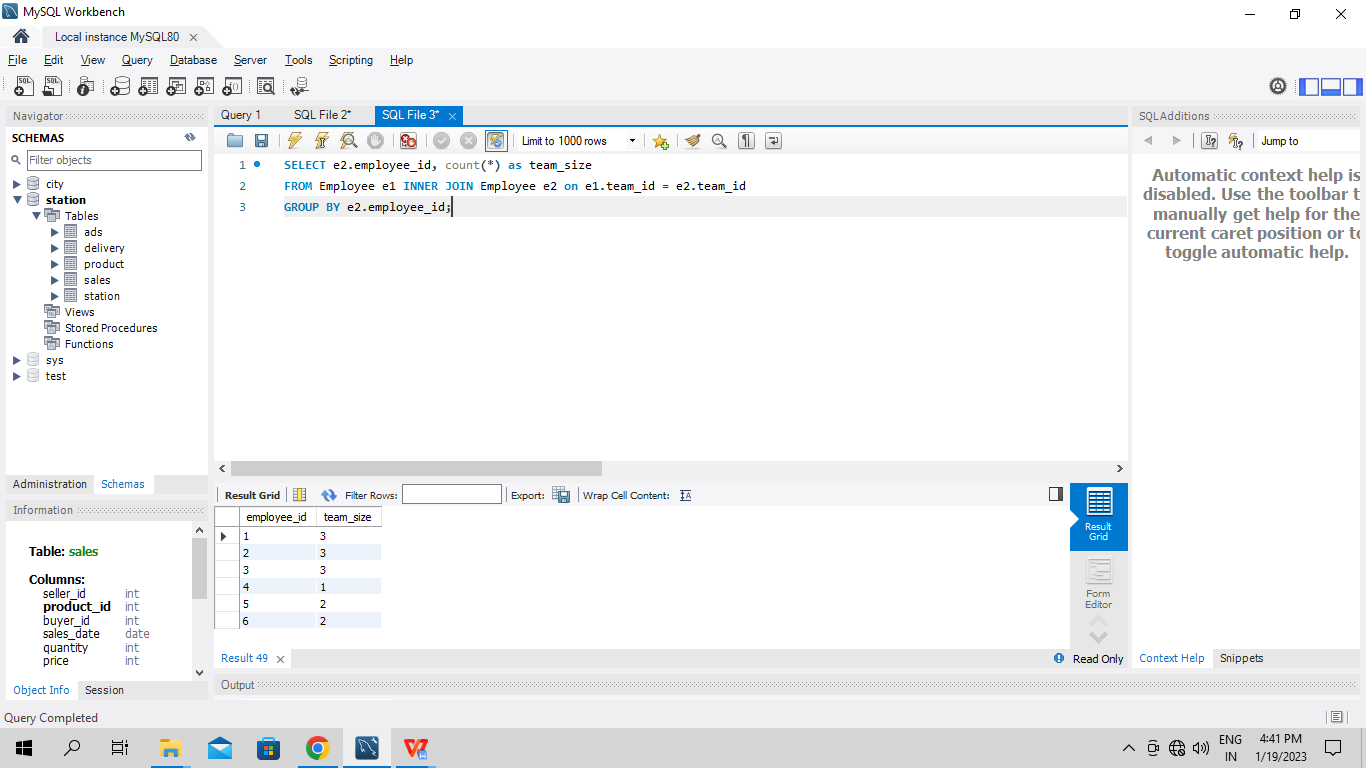


**Q21. Write an SQL query to find the team size of each of the employees. Return result table in any order.**

SELECT e2.employee\_id, count(\*) as team\_size

FROM Employee e1 INNER JOIN Employee e2 on e1.team\_id = e2.team\_id

GROUP BY e2.employee\_id;



**Q22. 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 DISTINCT(country\_name),

CASE

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

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

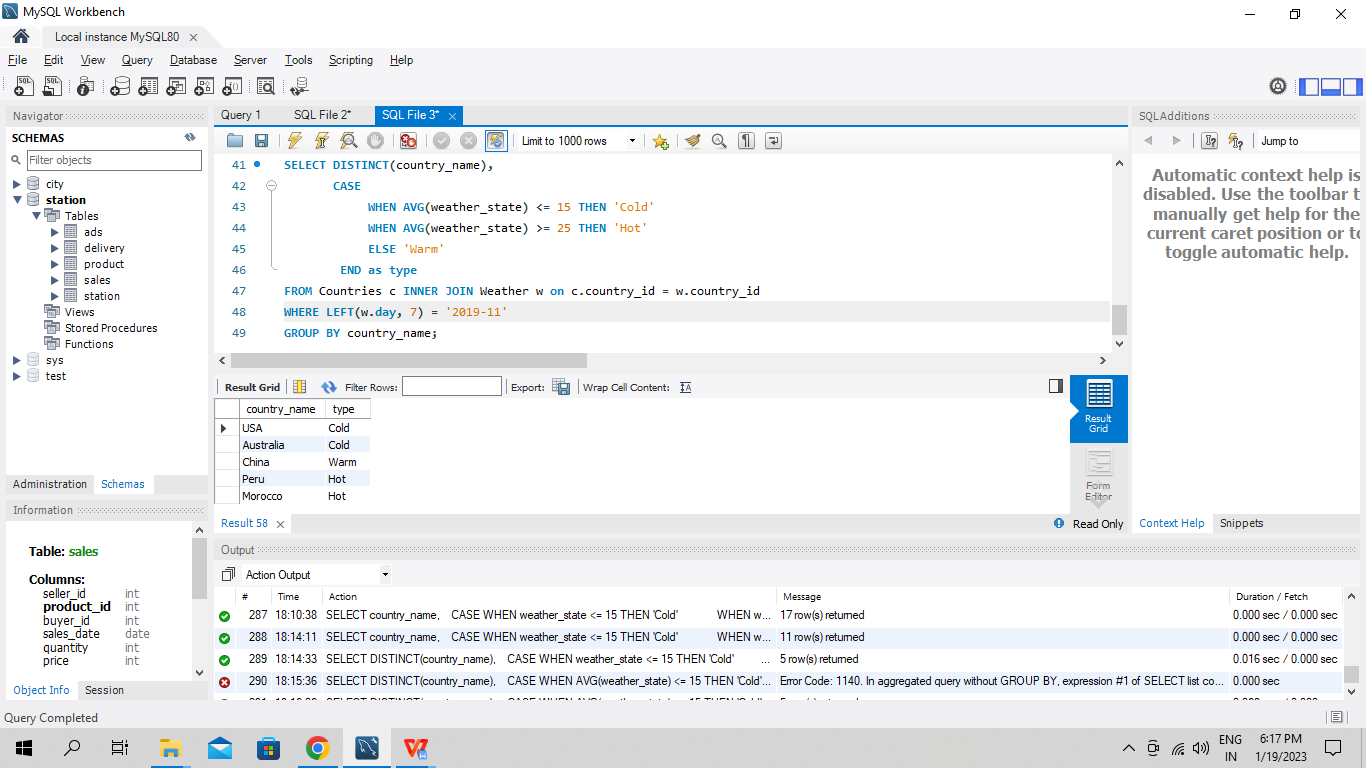
ELSE 'Warm'

END as type

FROM Countries c INNER JOIN Weather w on c.country\_id = w.country\_id

WHERE LEFT(w.day, 7) = '2019-11'

GROUP BY country\_name;



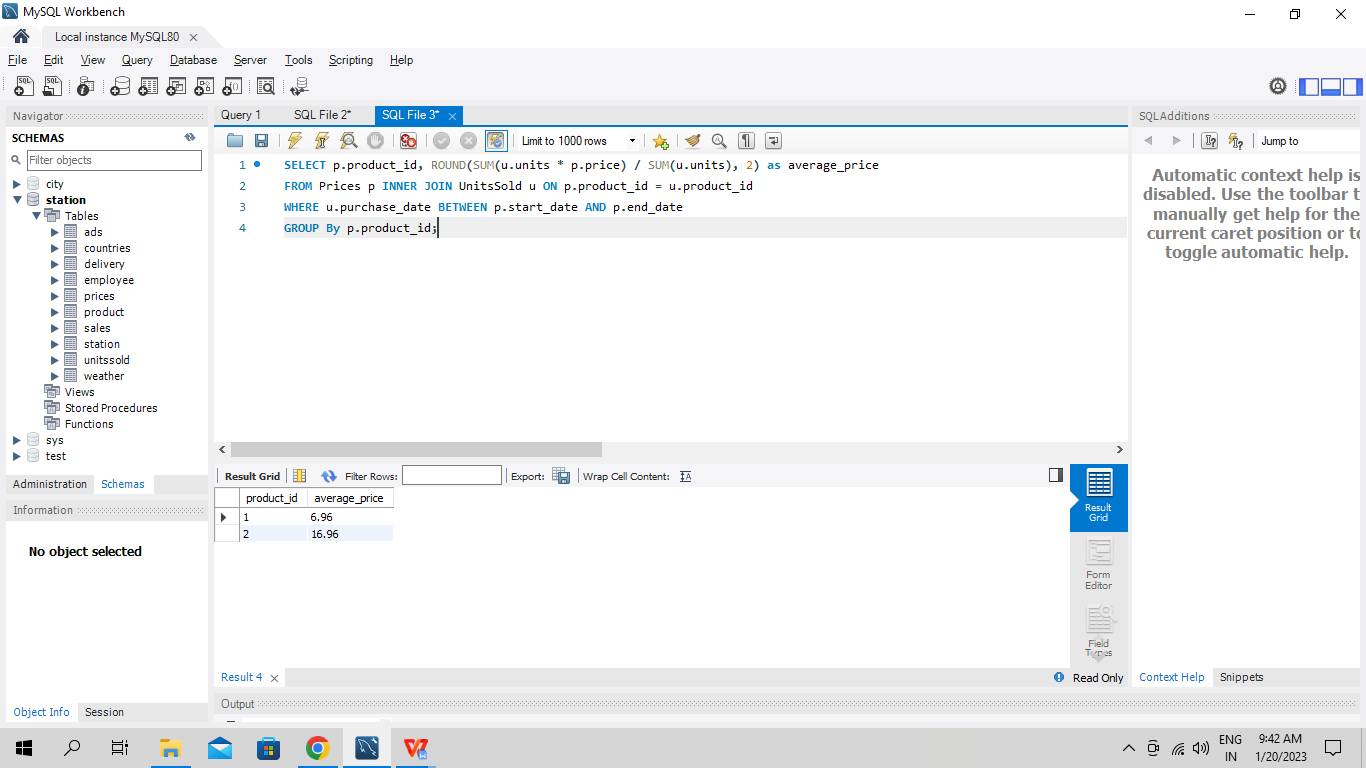
**Q23. Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places. Return the result table in any order.**

SELECT p.product\_id, ROUND(SUM(u.units \* p.price) / SUM(u.units), 2) as average\_price

FROM Prices p INNER JOIN UnitsSold u ON p.product\_id = u.product\_id

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

GROUP By p.product\_id;

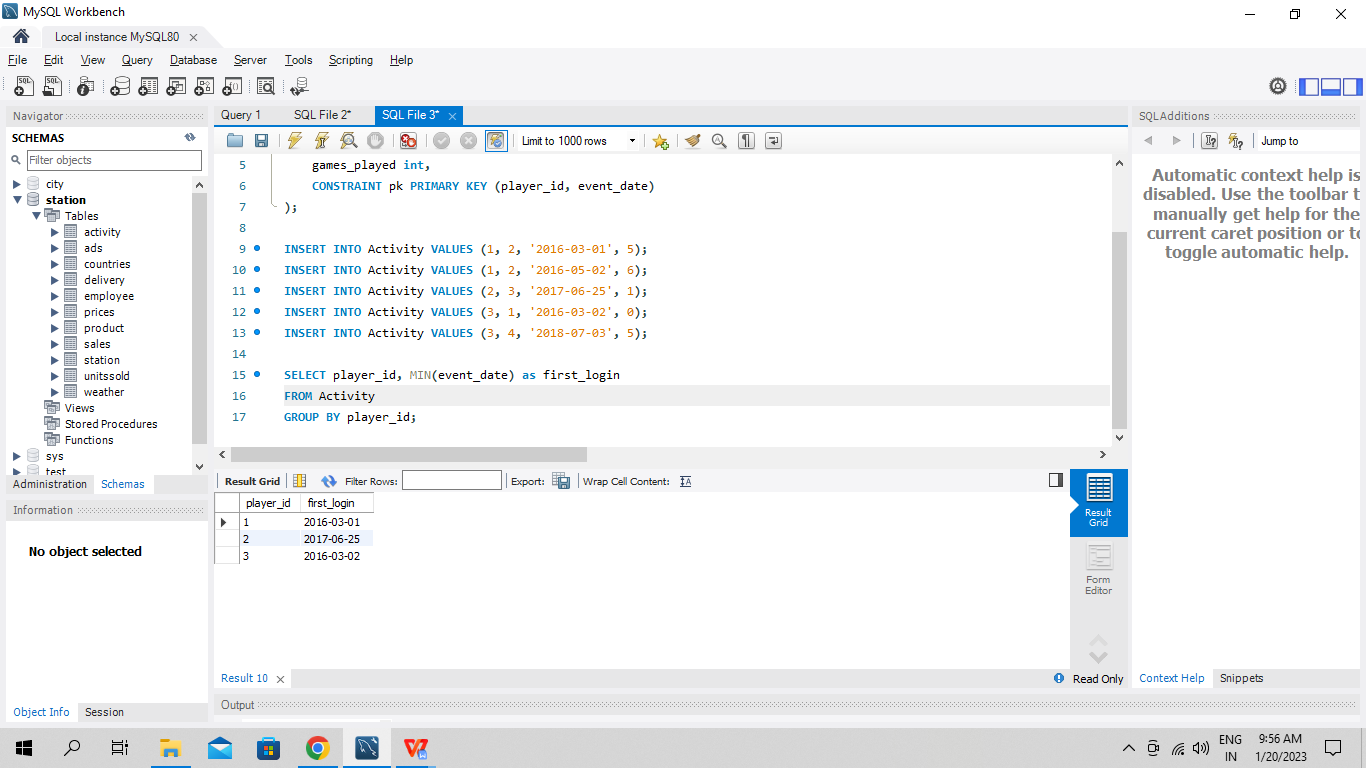


**Q24. Write an SQL query to report the first login date for each player. Return the result table in any order.**

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.**

SELECT player\_id, device\_id

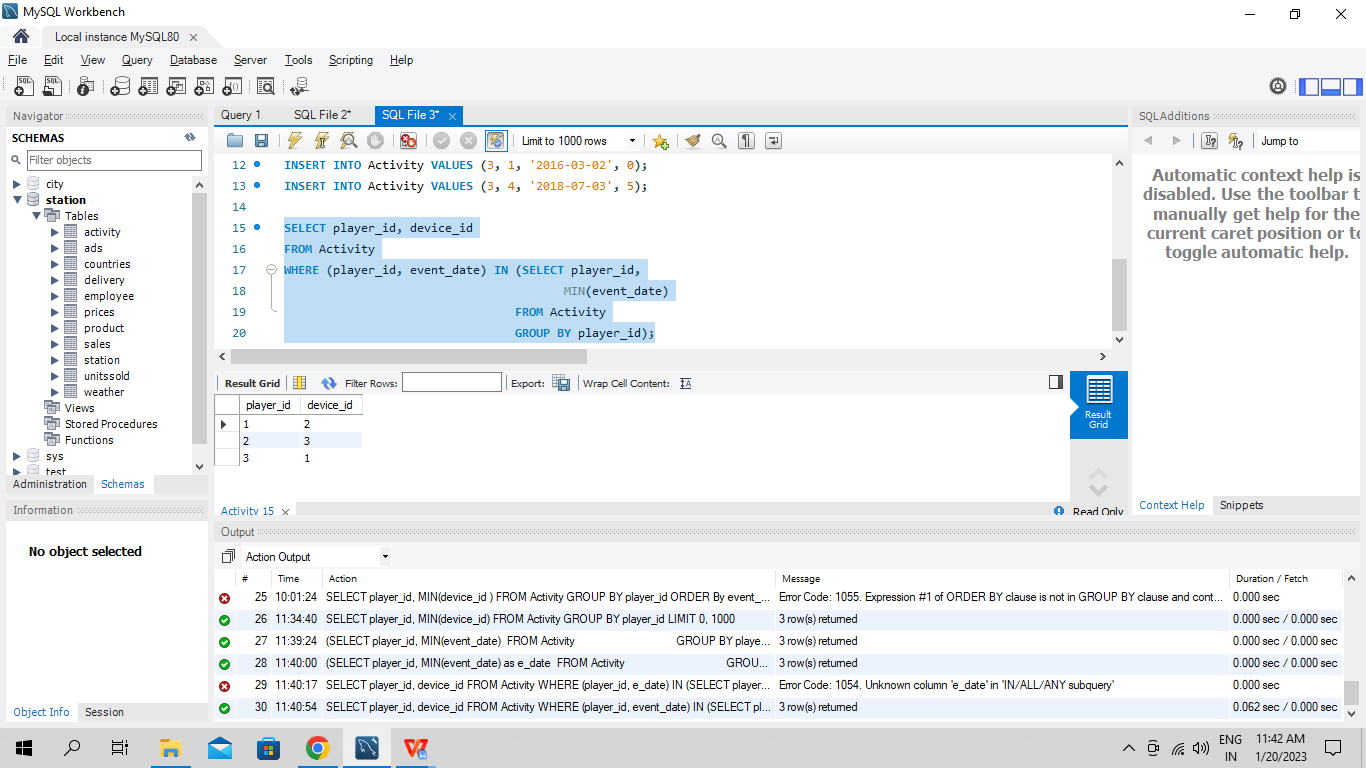
FROM Activity

WHERE (player\_id, event\_date) IN (SELECT player\_id,

MIN(event\_date)

FROM Activity

GROUP BY player\_id);



**Q26. 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.**

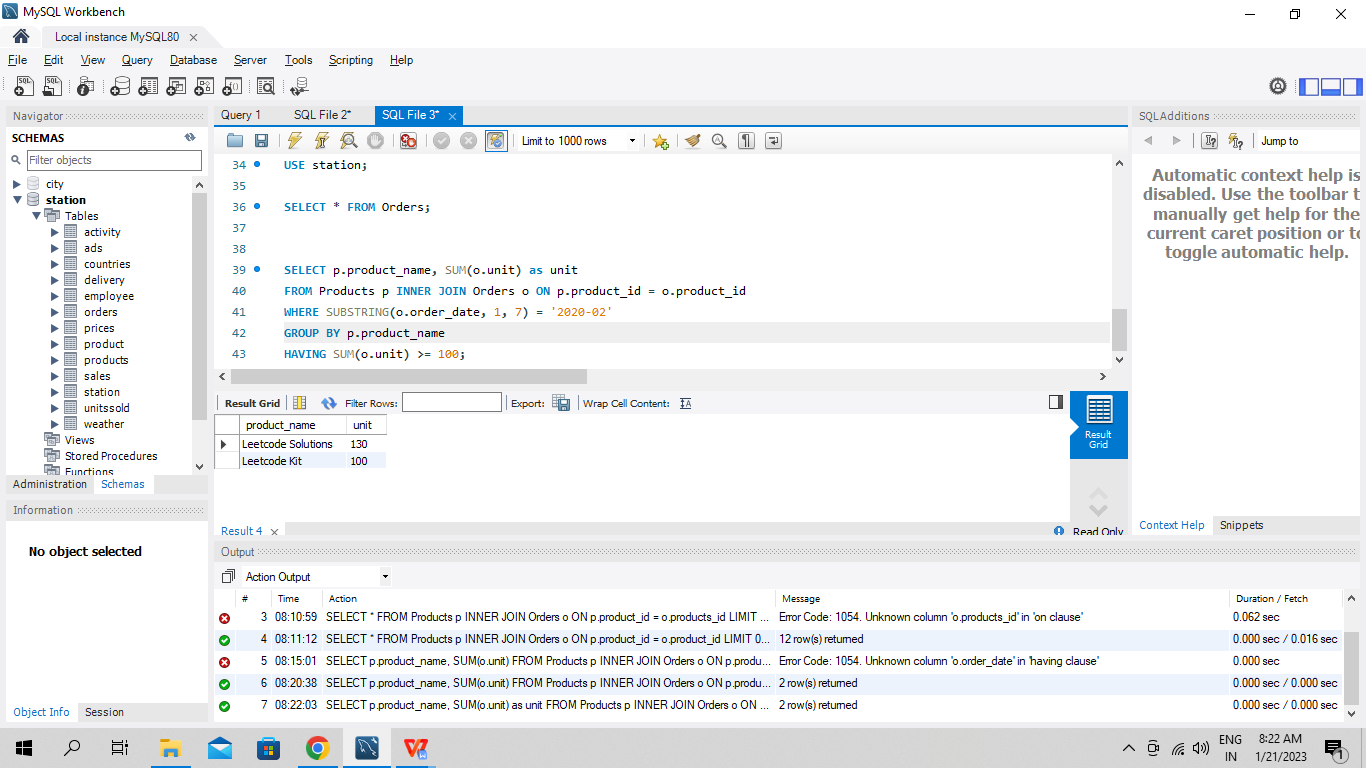
SELECT p.product\_name, SUM(o.unit) as unit

FROM Products p INNER JOIN Orders o ON p.product\_id = o.product\_id

WHERE SUBSTRING(o.order\_date, 1, 7) = '2020-02'

GROUP BY p.product\_name

HAVING SUM(o.unit) >= 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:**

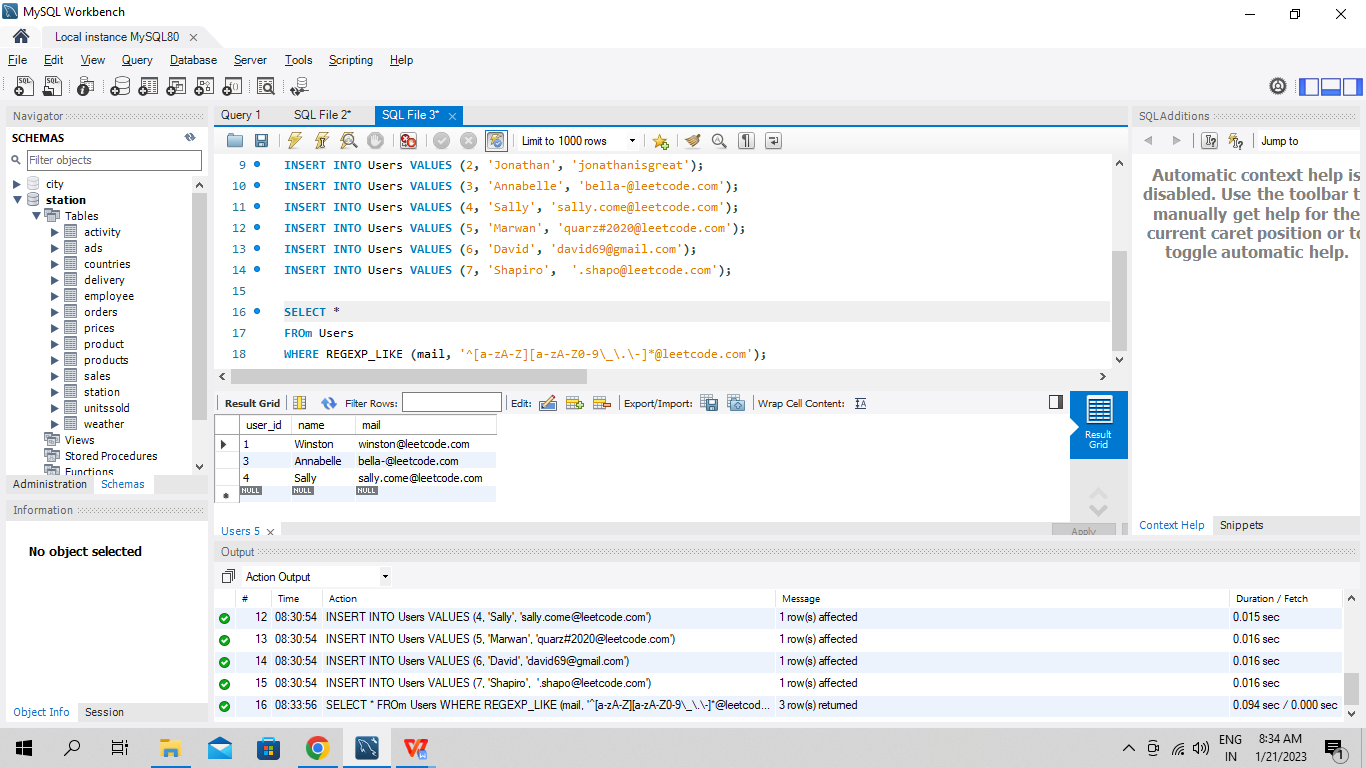
**● The prefix name is a string that may contain letters (upper or lower case), digits, underscore '\_', period '.', and/or dash '-'. The prefix name must start with a letter.**

**● The domain is '@leetcode.com'. Return the result table in any order.**

SELECT \*

FROm Users

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



**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. 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

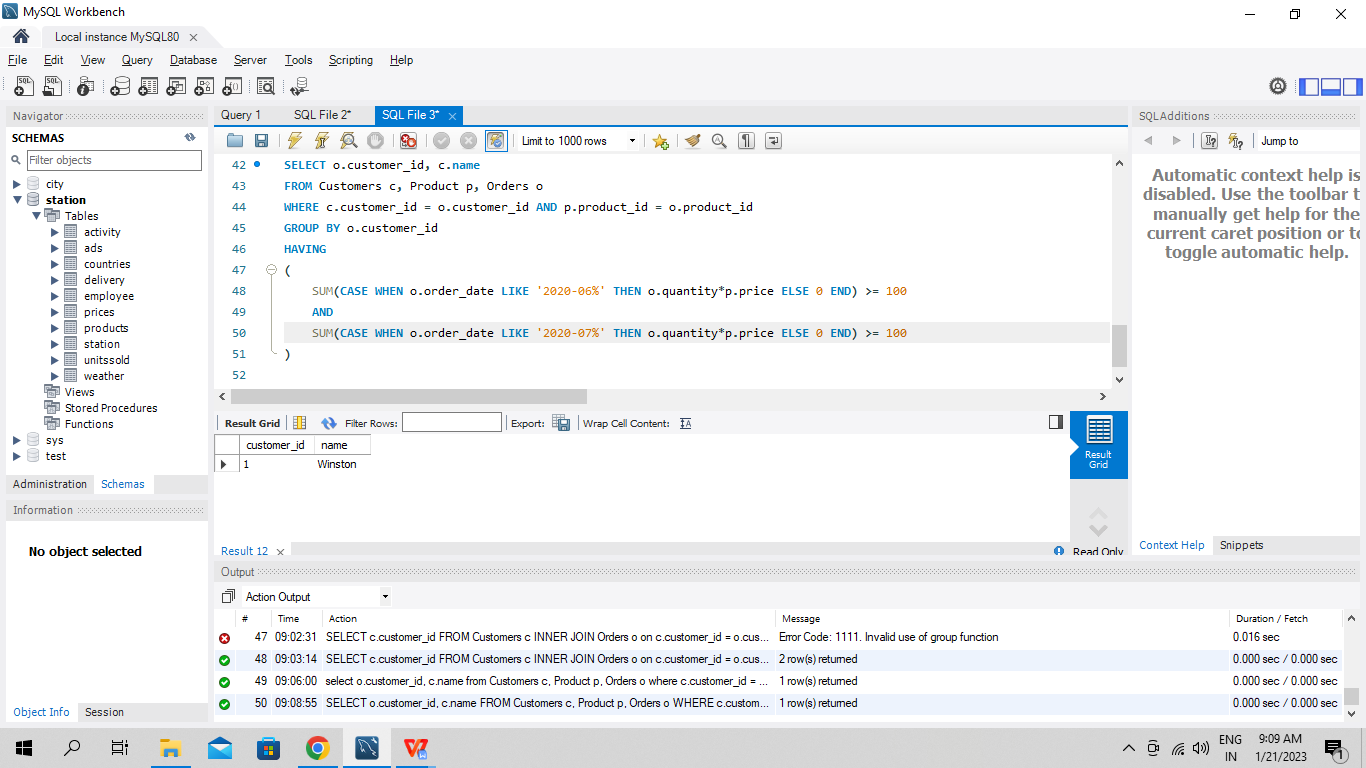
(

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

)

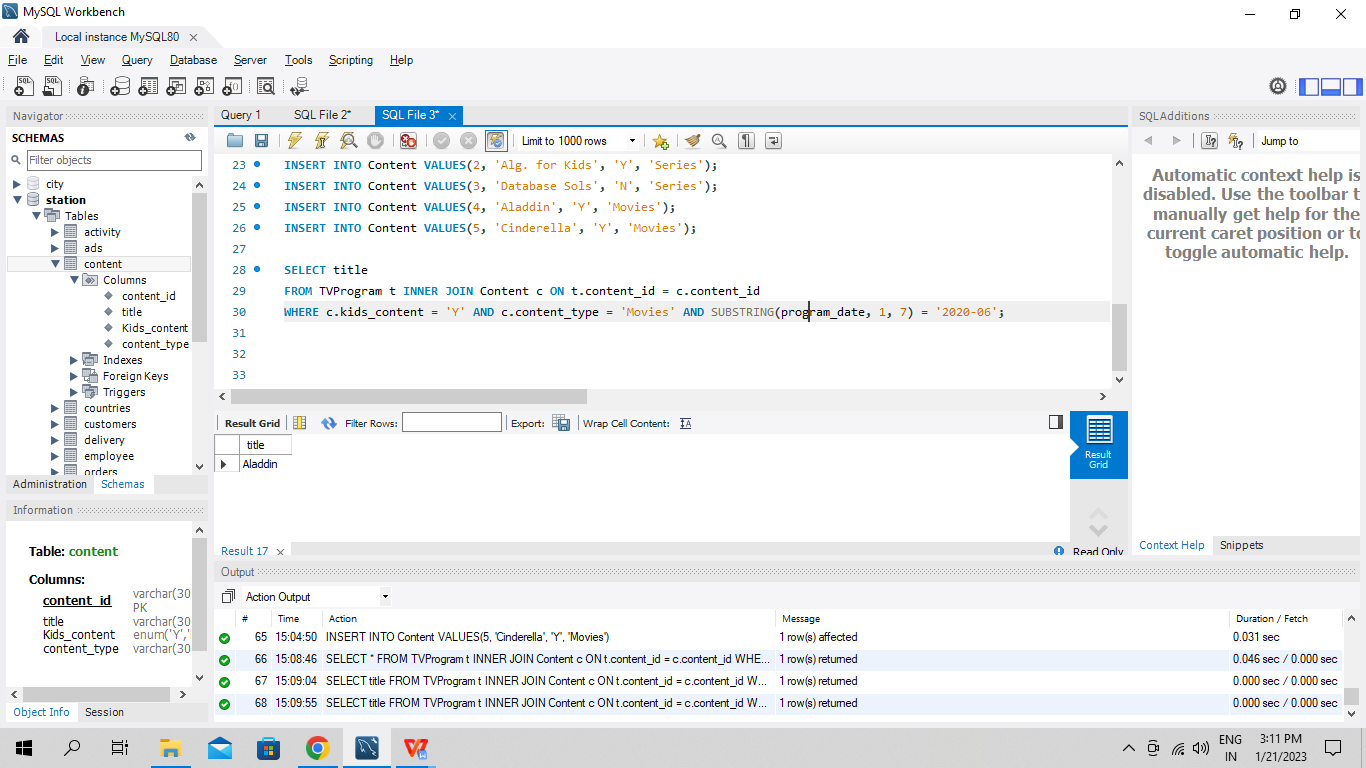


**Q29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020. Return the result table in any order**.

SELECT title

FROM TVProgram t INNER JOIN Content c ON t.content\_id = c.content\_id

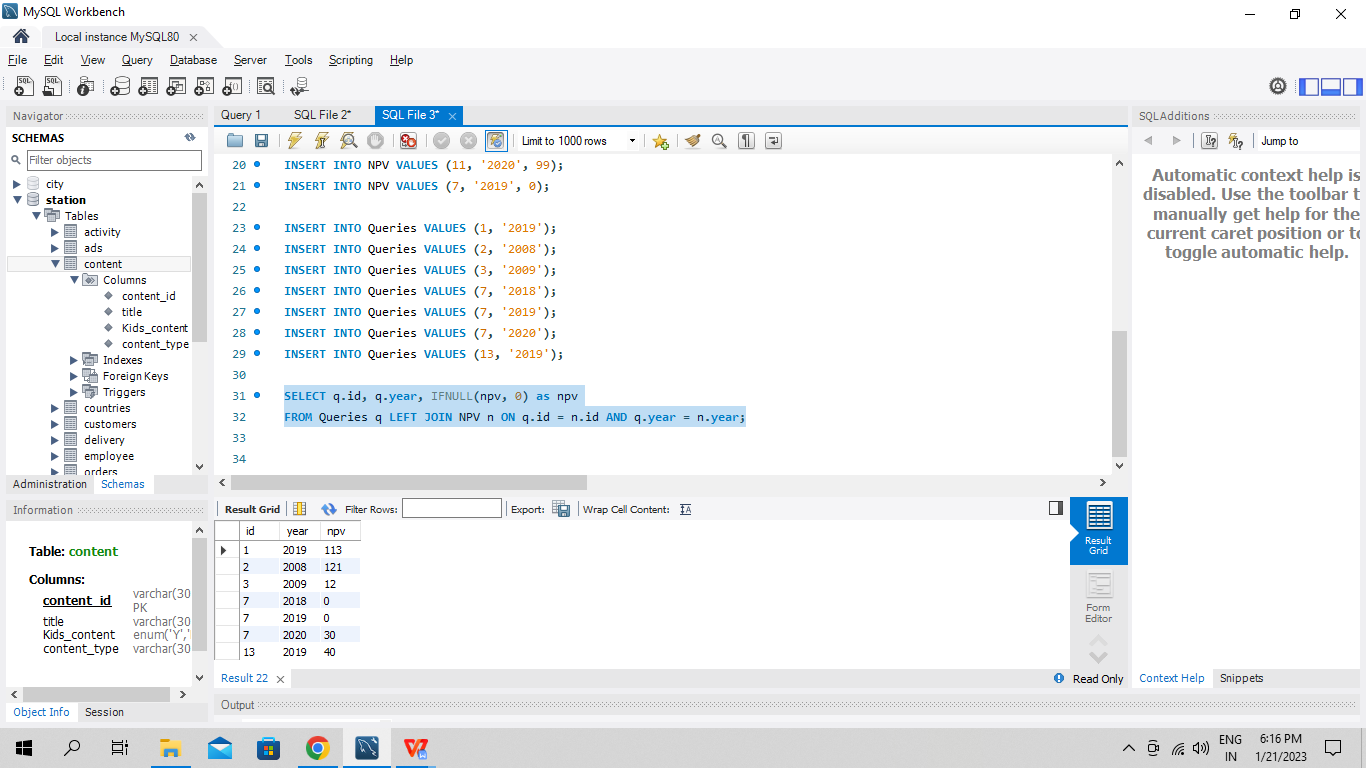
WHERE c.kids\_content = 'Y' AND c.content\_type = 'Movies' AND SUBSTRING(program\_date, 1, 7) = '2020-06';



**Q30. 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, IFNULL(npv, 0) as npv

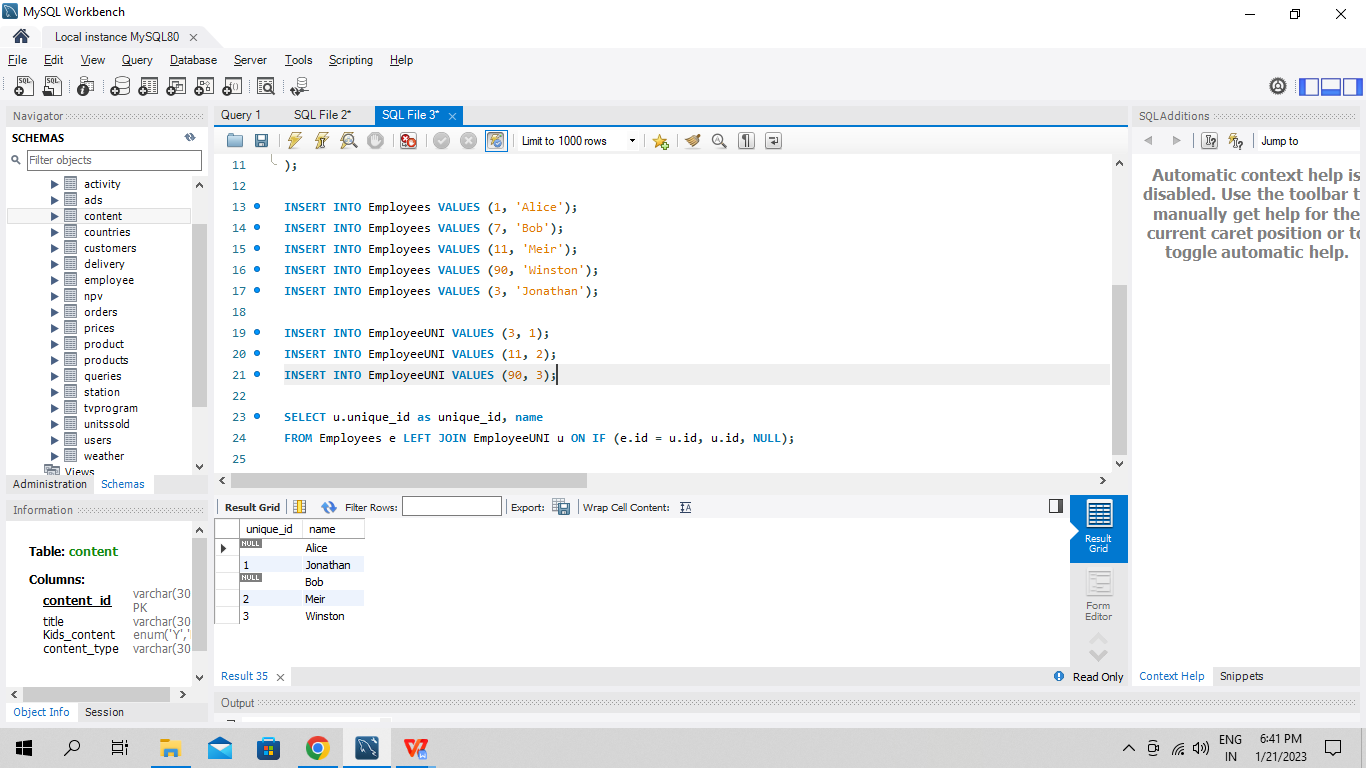
FROM Queries q LEFT JOIN NPV n ON q.id = n.id AND q.year = n.year;



**Q31. 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 u.unique\_id as unique\_id, name

FROM Employees e LEFT JOIN EmployeeUNI u ON IF (e.id = u.id, u.id, NULL);



**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**

SELECT tmp.name, tmp.distance

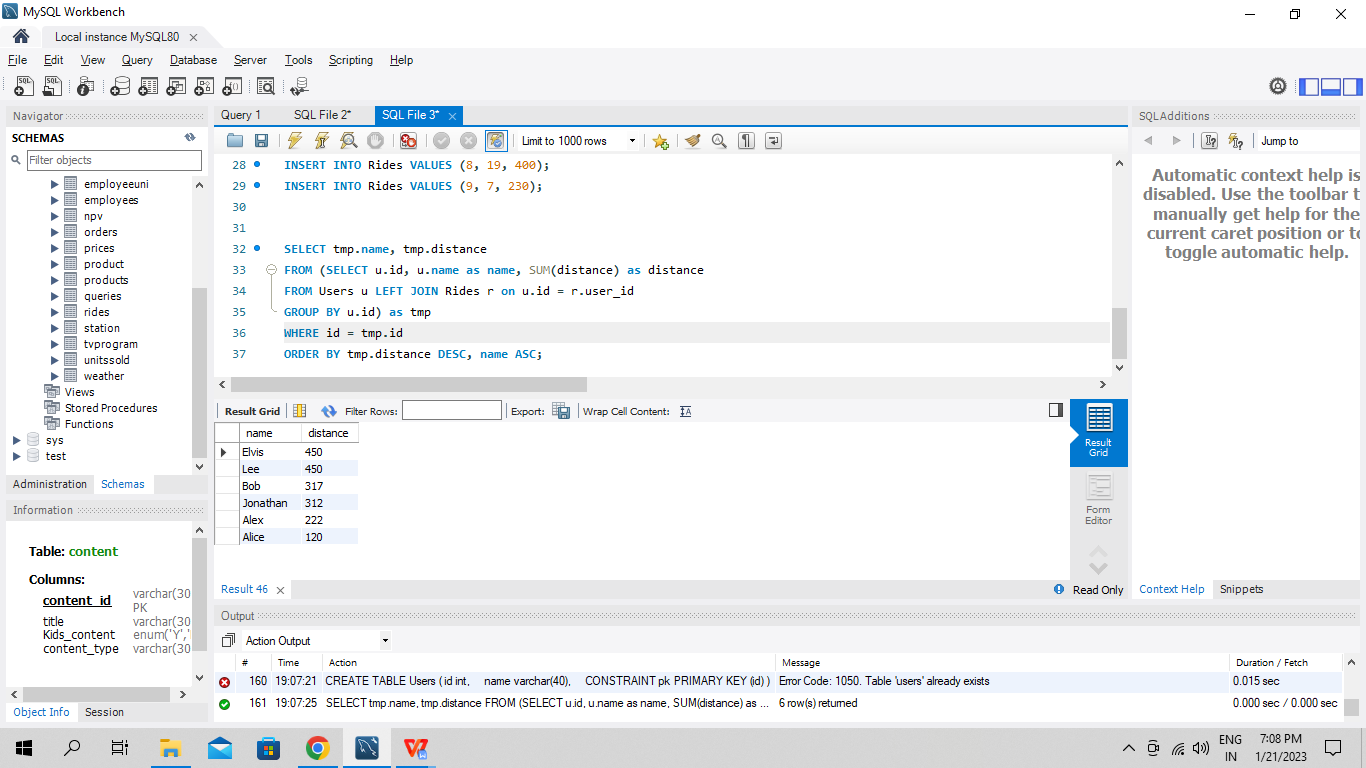
FROM (SELECT u.id, u.name as name, SUM(distance) as distance

FROM Users u LEFT JOIN Rides r on u.id = r.user\_id

GROUP BY u.id) as tmp

WHERE id = tmp.id

ORDER BY tmp.distance DESC, name ASC;



**Q34. 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 first\_query.name as results

FROM (SELECT u.name, COUNT(\*) as count

FROM Users u INNER JOIN MovieRating m ON u.user\_id = m.user\_id

GROUP BY u.user\_id

ORDER BY count DESC, u.name ASC LIMIT 1) first\_query

UNION

SELECT second\_query.title as ratings

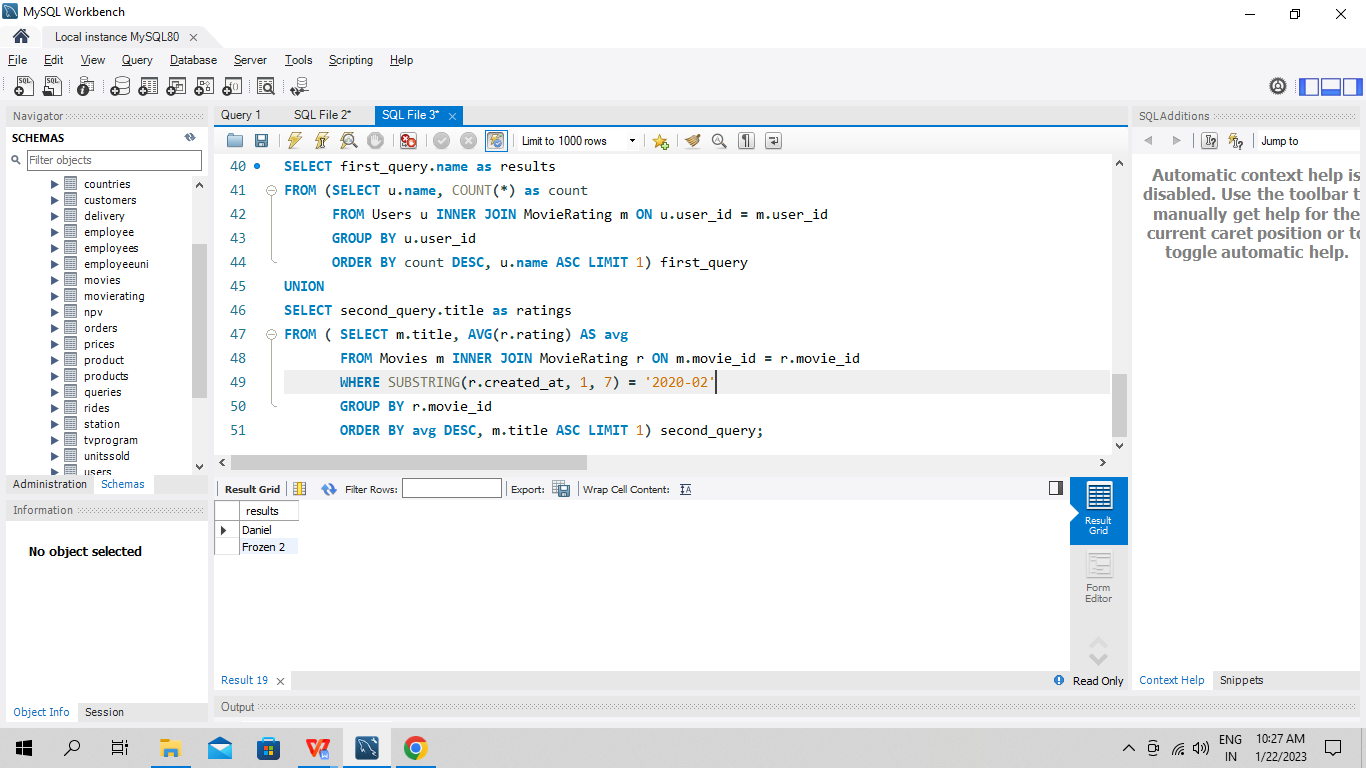
FROM ( SELECT m.title, AVG(r.rating) AS avg

FROM Movies m INNER JOIN MovieRating r ON m.movie\_id = r.movie\_id

WHERE SUBSTRING(r.created\_at, 1, 7) = '2020-02'

GROUP BY r.movie\_id

ORDER BY avg DESC, m.title ASC LIMIT 1) second\_query;



**Q35. 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 id, name

FROM Students

WHERE department\_id NOT IN (SELECT id

FROM Deparments);

**Q36. 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

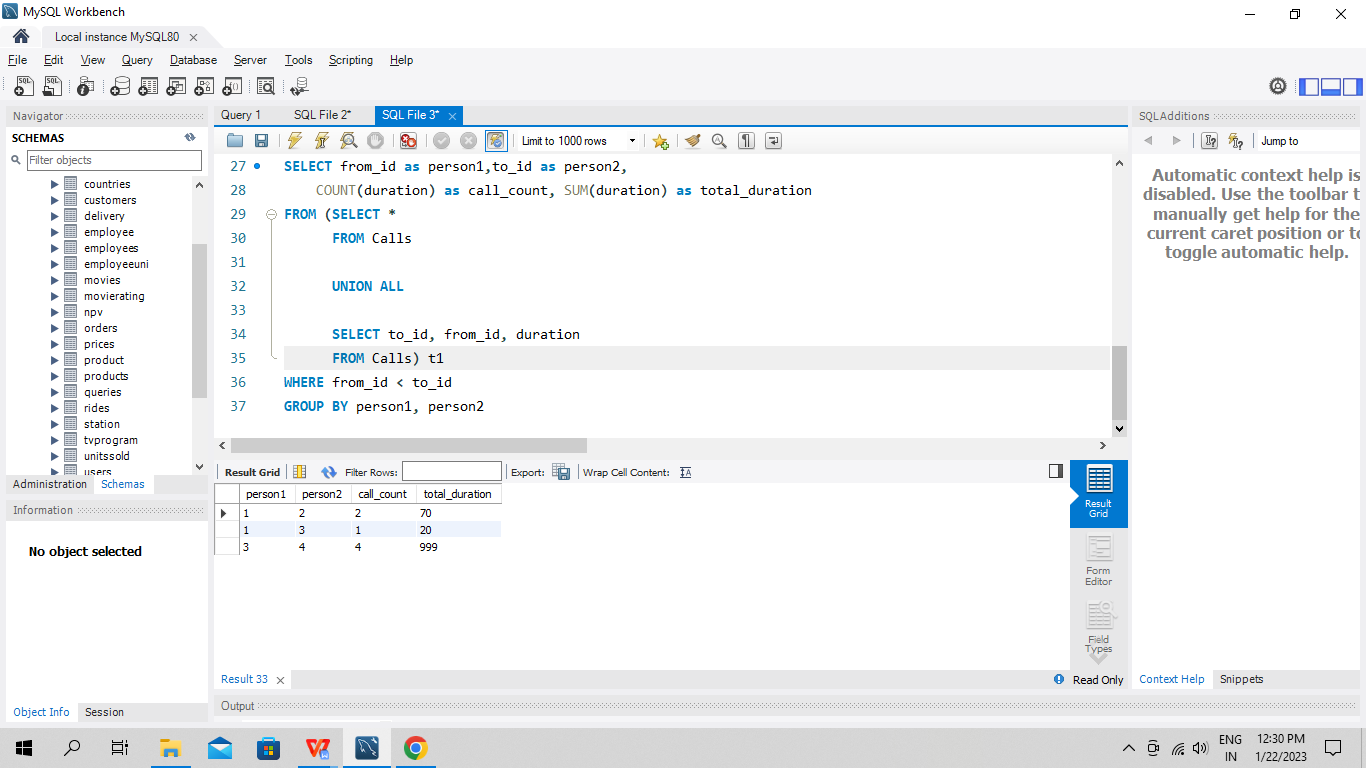
UNION ALL

SELECT to\_id, from\_id, duration

FROM Calls) t1

WHERE from\_id < to\_id

GROUP BY person1, person2



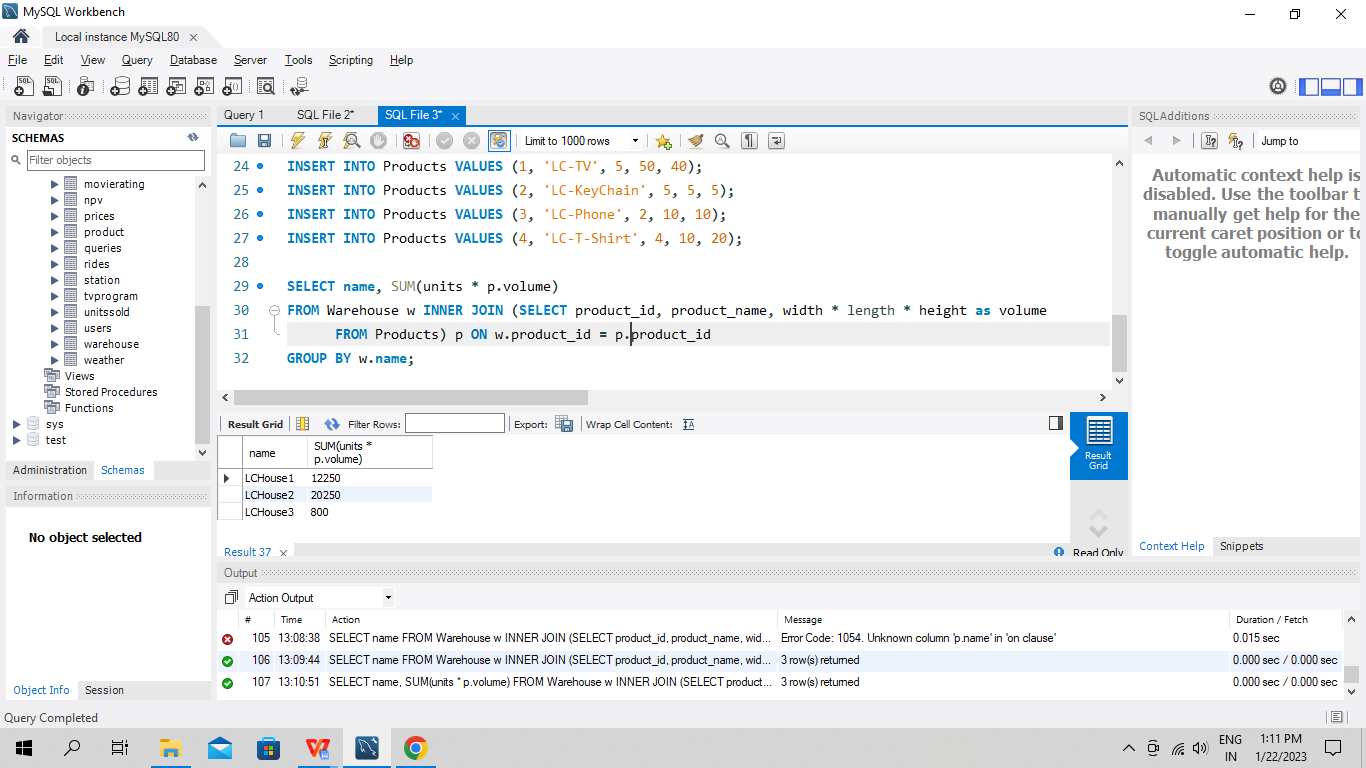
**Q37. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse.**

SELECT name, SUM(units \* p.volume)

FROM Warehouse w INNER JOIN (SELECT product\_id, product\_name, width \* length \* height as volume

FROM Products) p ON w.product\_id = p.product\_id

GROUP BY w.name;



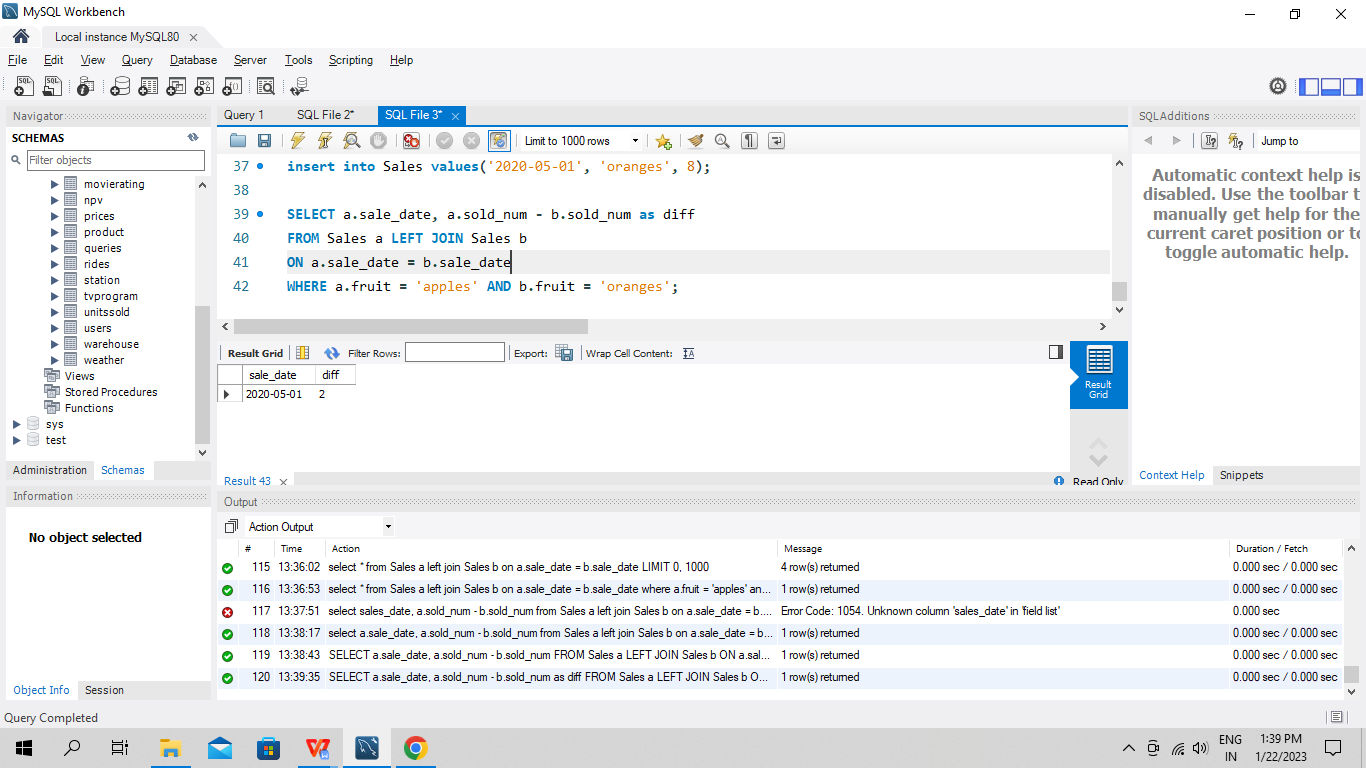
**Q38. Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale\_date.**

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';



**Q39. 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 first\_date AS (

SELECT player\_id, MIN(event\_date) as firstdate

FROM Activity

GROUP BY player\_id

)

SELECT SUM(temp.case\_col) / COUNT(DISTINCT(temp.player\_id)) as fraction

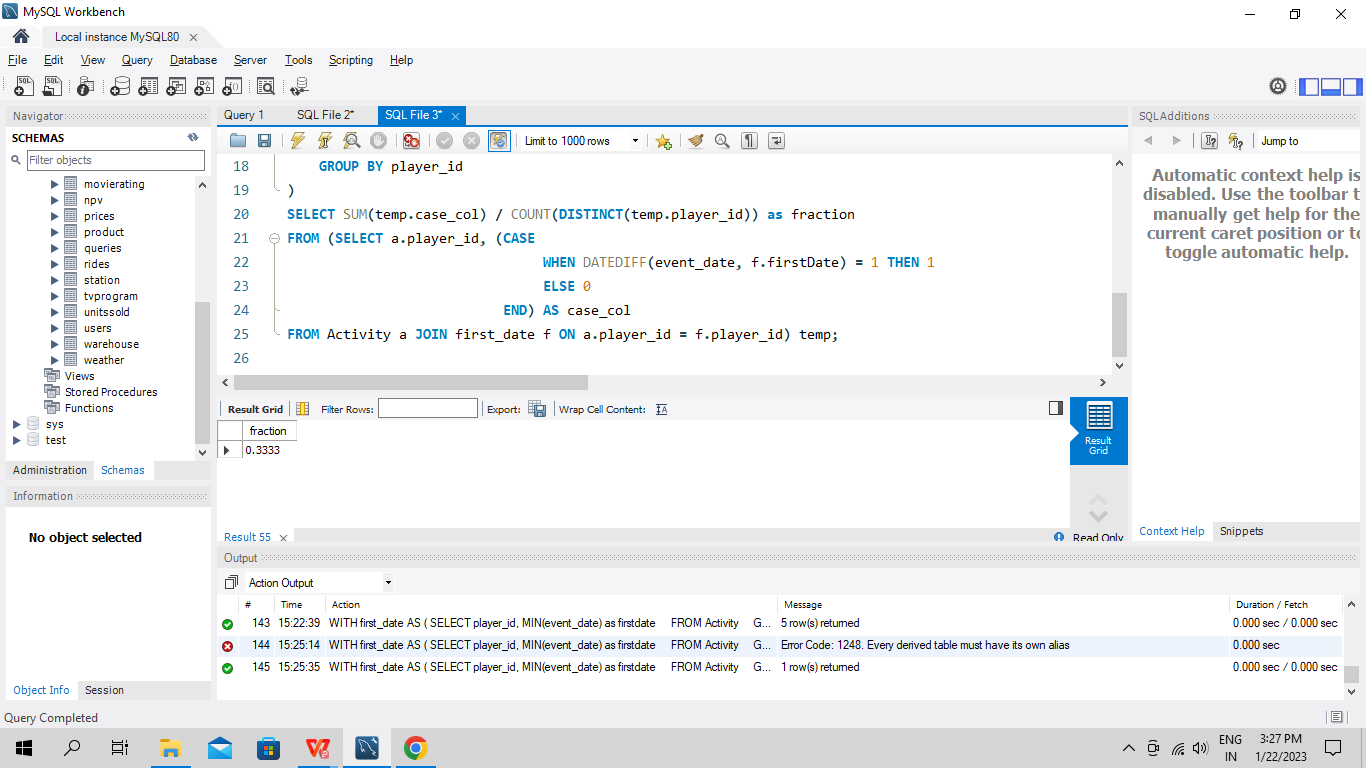
FROM (SELECT a.player\_id, (CASE

WHEN DATEDIFF(event\_date, f.firstDate) = 1 THEN 1

ELSE 0

END) AS case\_col

FROM Activity a JOIN first\_date f ON a.player\_id = f.player\_id) temp;



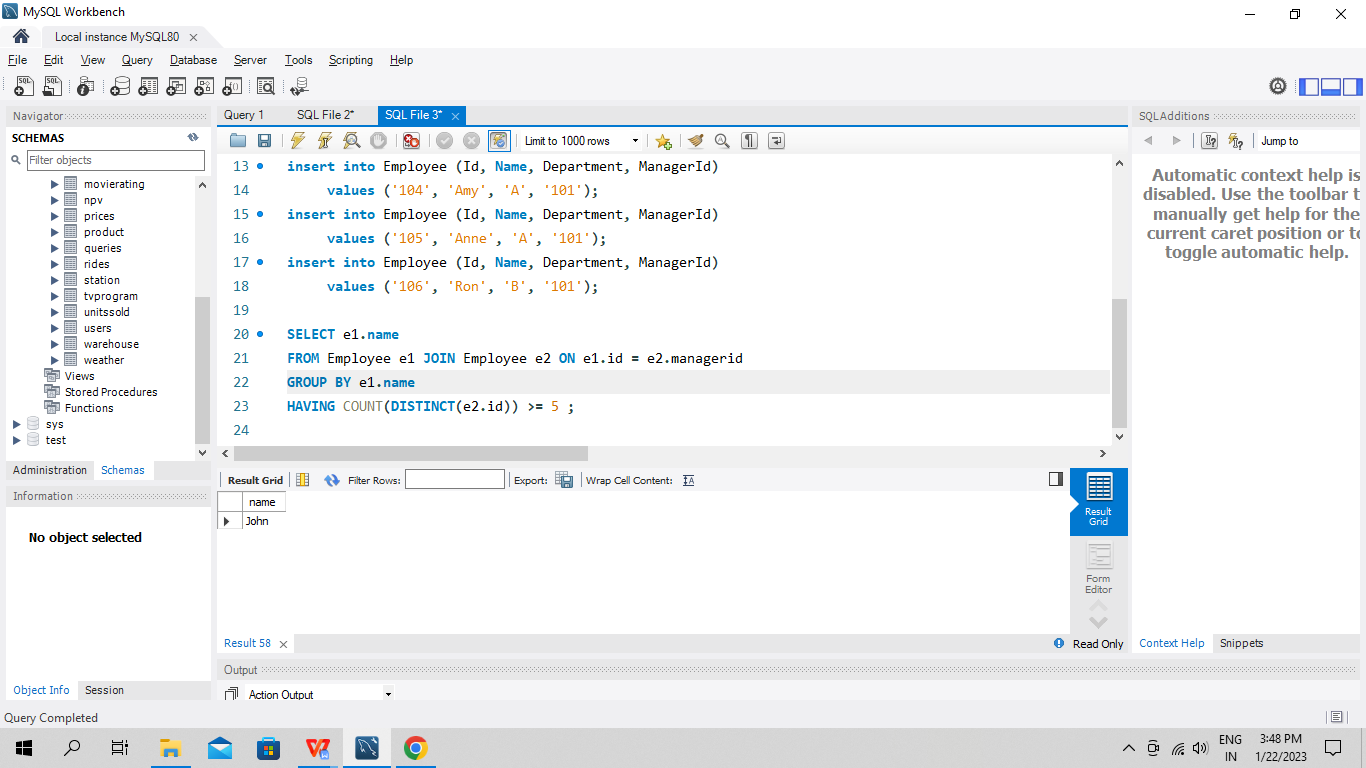
**Q40. Write an SQL query to report the managers with at least five direct reports.**

SELECT e1.name

FROM Employee e1 JOIN Employee e2 ON e1.id = e2.managerid

GROUP BY e1.name

HAVING COUNT(DISTINCT(e2.id)) >= 5 ;

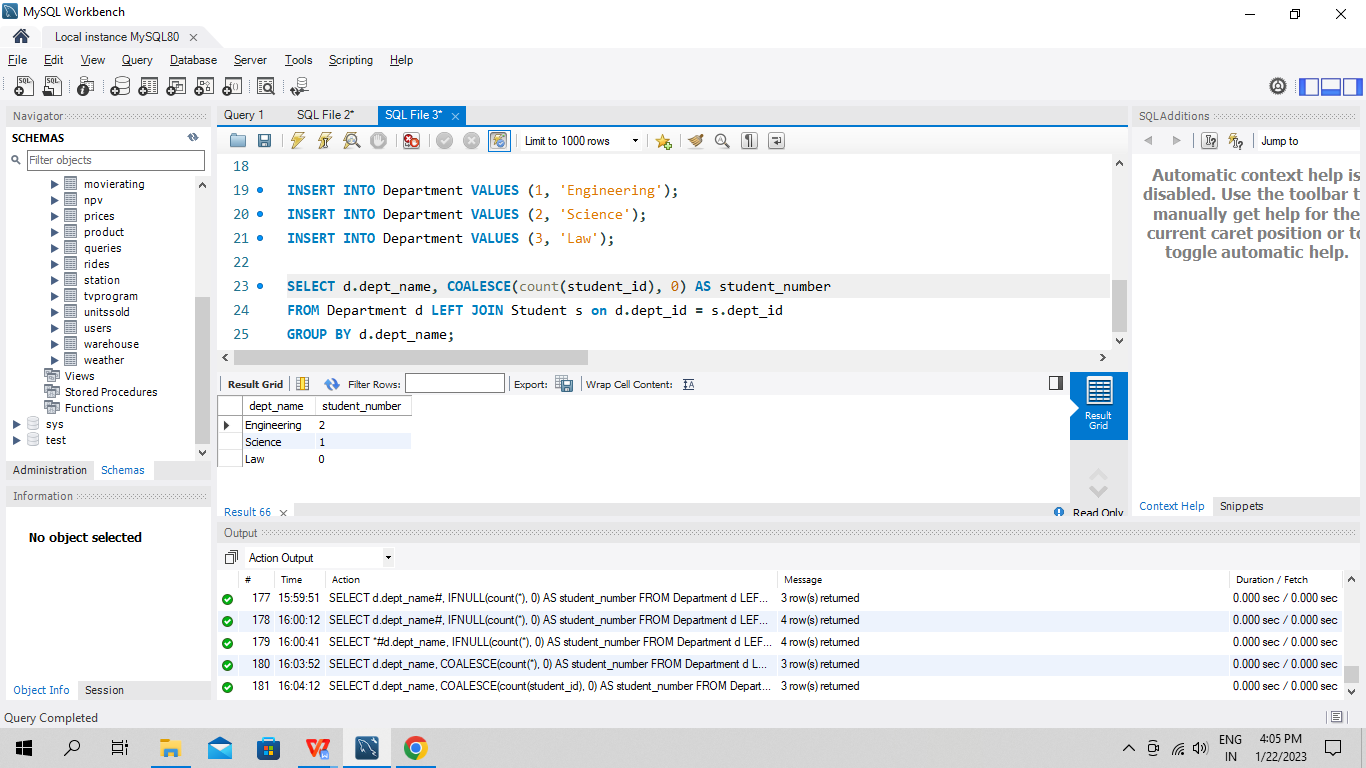


**Q41. Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students). Return the result table ordered by student\_number in descending order. In case of a tie, order them by dept\_name alphabetically.**

SELECT d.dept\_name, COALESCE(count(student\_id), 0) AS student\_number

FROM Department d LEFT JOIN Student s on d.dept\_id = s.dept\_id

GROUP BY d.dept\_name;



**Q42. 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 \*

FROM Product)

**Q43. 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

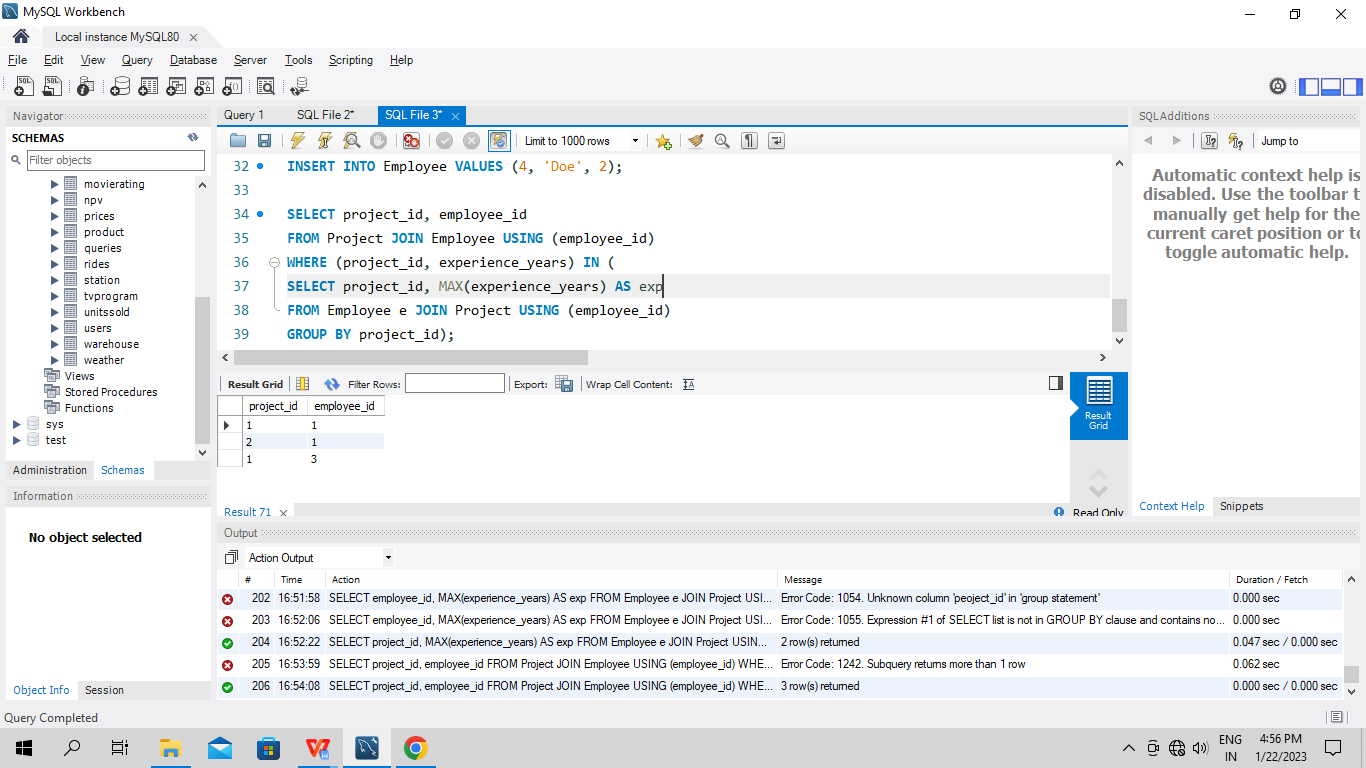
FROM Project JOIN Employee USING (employee\_id)

WHERE (project\_id, experience\_years) IN (

SELECT project\_id, MAX(experience\_years) AS exp

FROM Employee e JOIN Project USING (employee\_id)

GROUP BY project\_id);



**Q44. 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 book\_id NOT IN ( SELECT book\_id

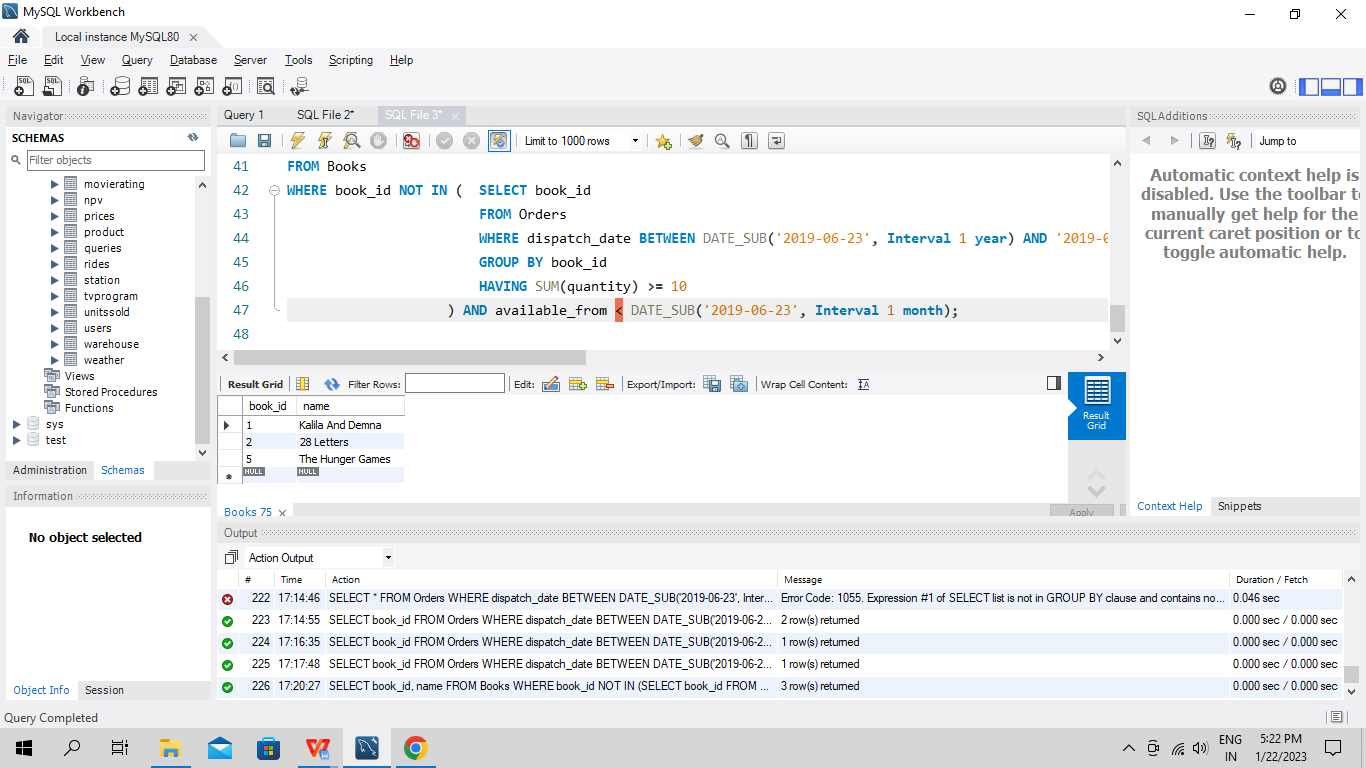
FROM Orders

WHERE dispatch\_date BETWEEN DATE\_SUB('2019-06-23', Interval 1 year) AND '2019-06-23'

GROUP BY book\_id

HAVING SUM(quantity) >= 10

) AND available\_from < DATE\_SUB('2019-06-23', Interval 1 month);



**Q45. 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.

**Q46. The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player\_id wins. Write an SQL query to find the winner in each group.**

SELECT group\_id, player\_id

FROM (SELECT group\_id, player\_id, SUM(score) AS score

FROM Players p INNER JOIN (SELECT first\_player AS player\_id, first\_score AS score

UNION ALL

SELECT second\_player AS player\_id, seconf\_score AS score) s ON p.player\_id = s.player\_id

GROUP BY p.player\_id

ORDER BY group\_id, score DESC, player\_id) top\_scores

GROUP BY group\_id;