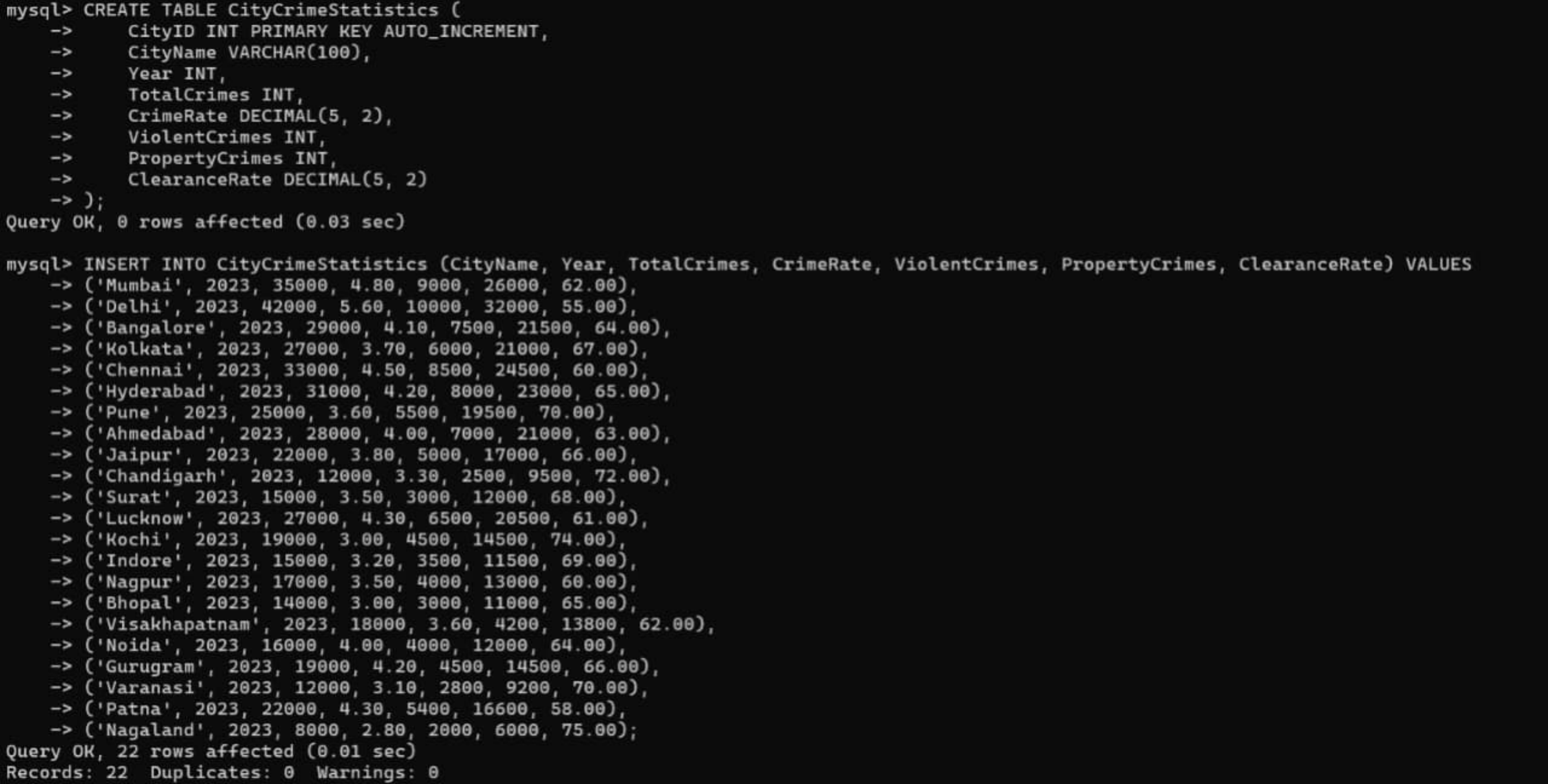
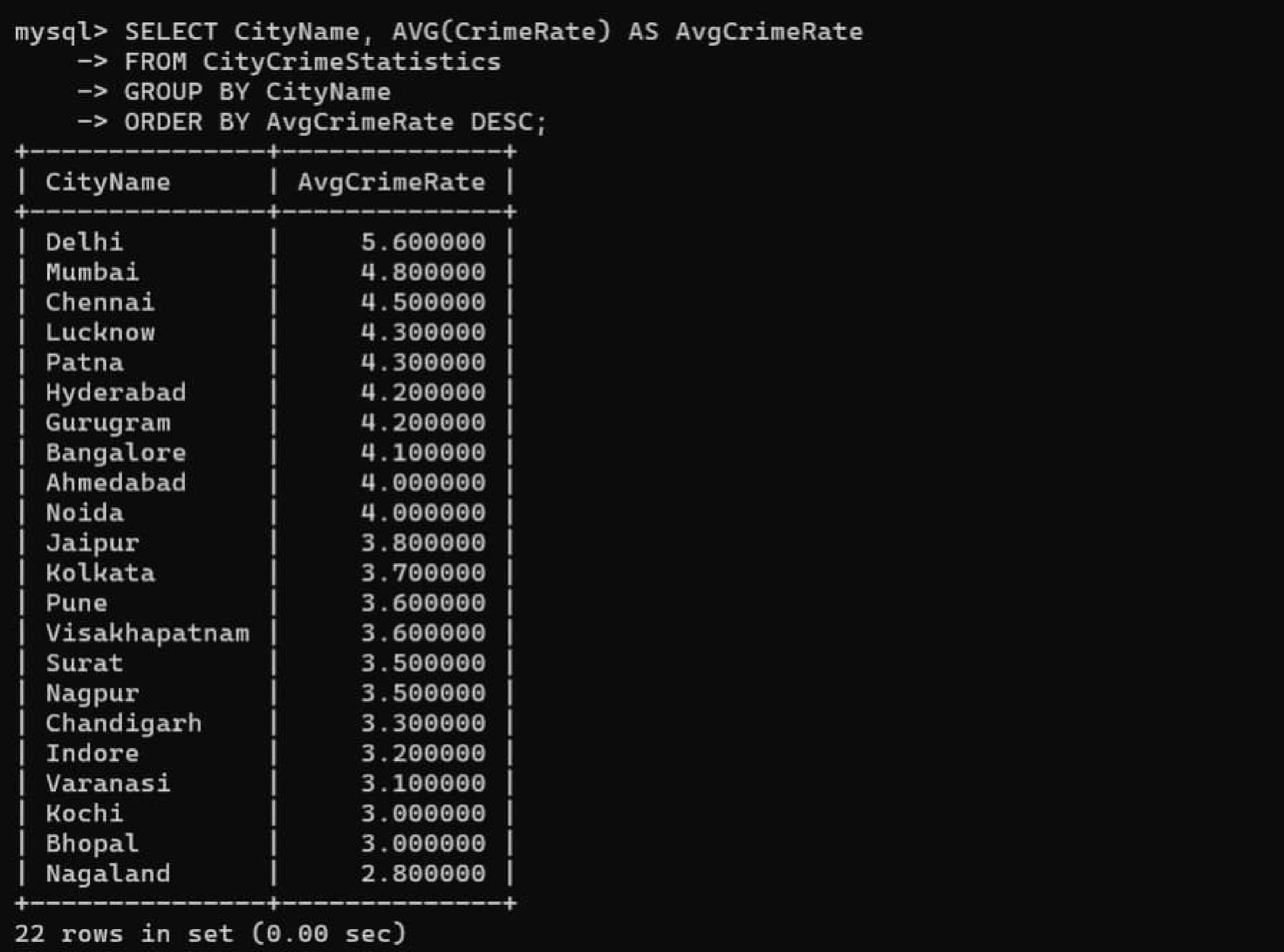
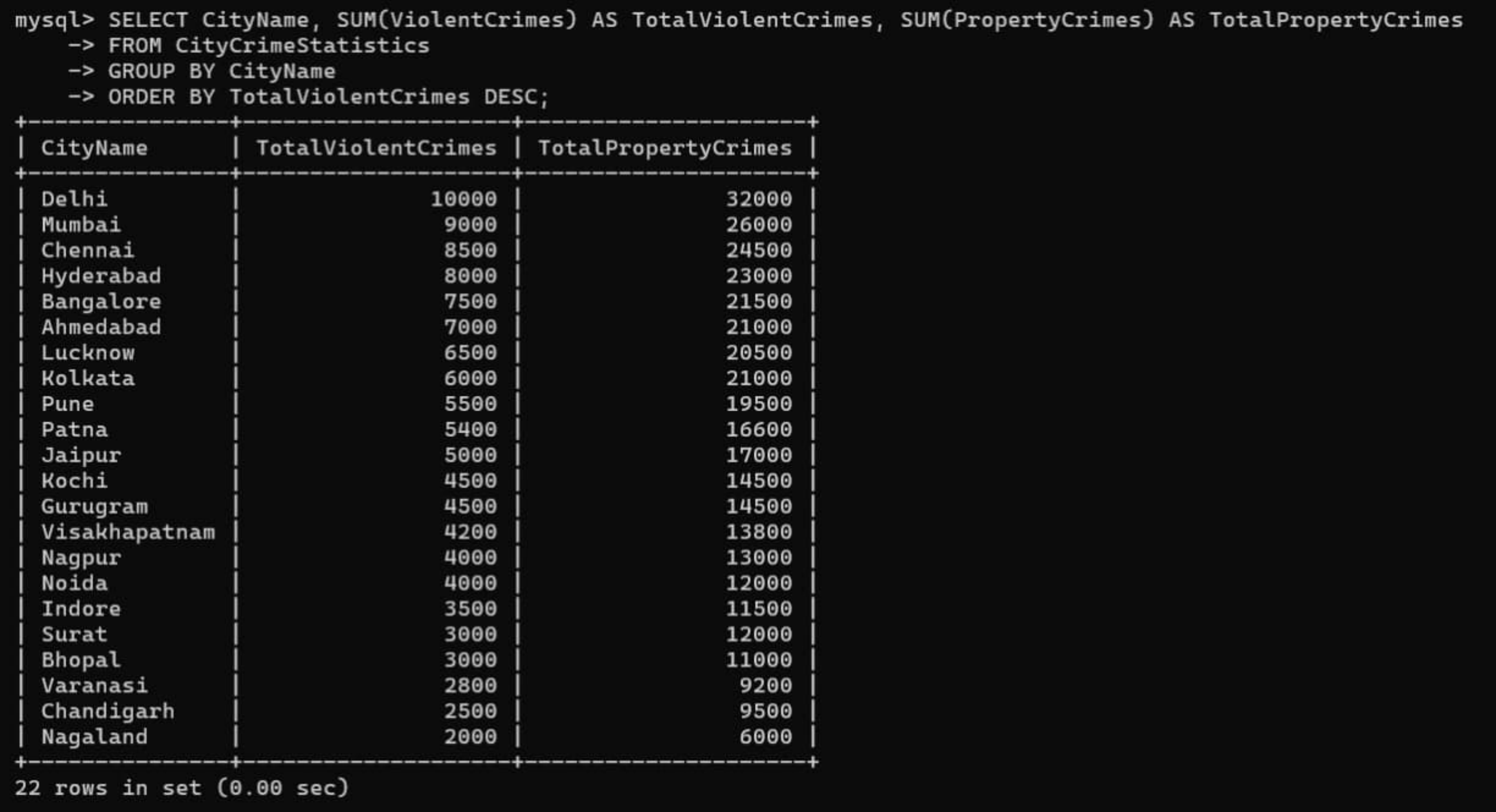


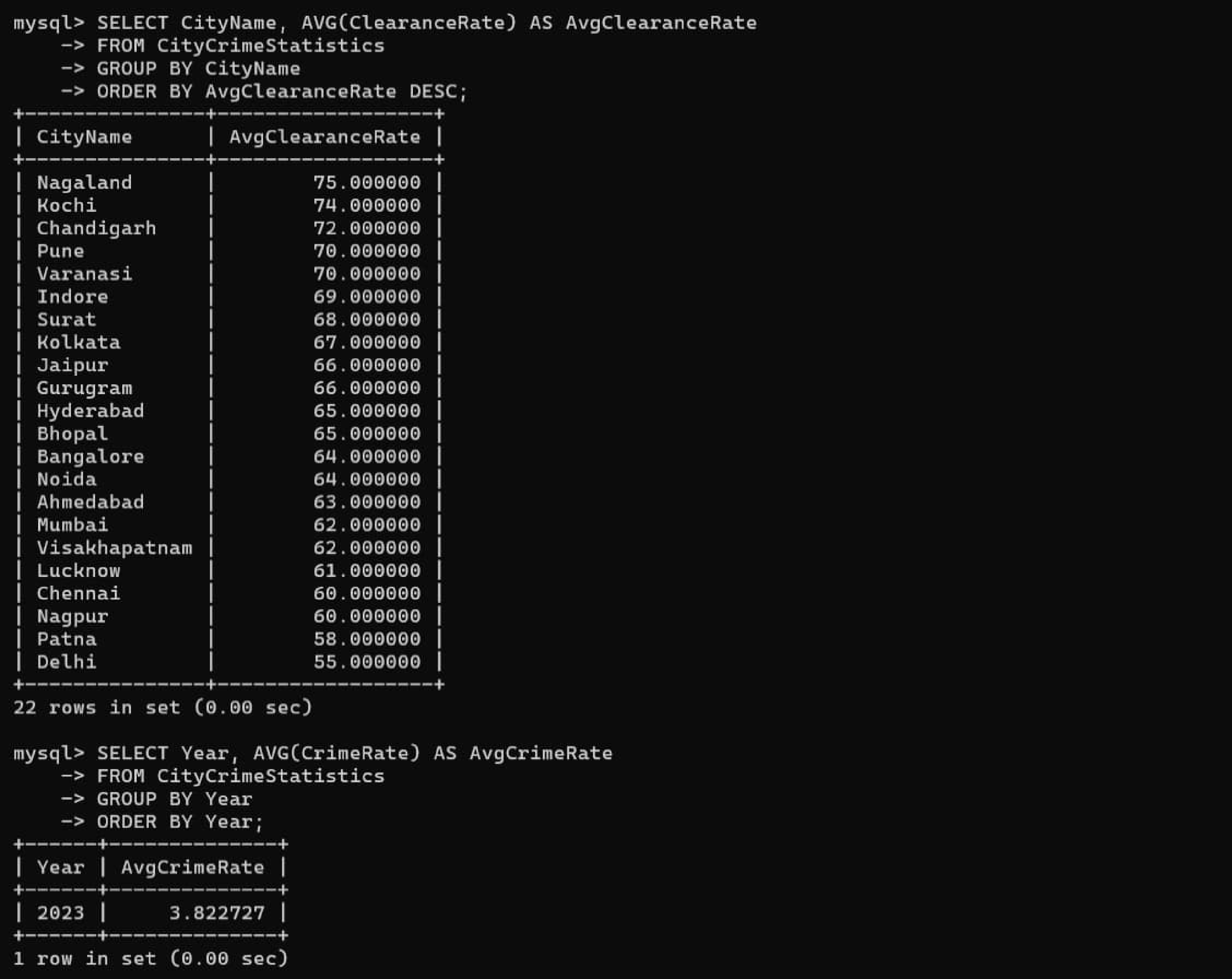
```
mysql> CREATE TABLE CityCrimeStatistics (
    ->     CityID INT PRIMARY KEY AUTO_INCREMENT,
    ->     CityName VARCHAR(100),
    ->     Year INT,
    ->     TotalCrimes INT,
    ->     CrimeRate DECIMAL(5, 2),
    ->     ViolentCrimes INT,
    ->     PropertyCrimes INT,
    ->     ClearanceRate DECIMAL(5, 2)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO CityCrimeStatistics (CityName, Year, TotalCrimes, CrimeRate, ViolentCrimes, PropertyCrimes, ClearanceRate) VALUES
    -> ('Mumbai', 2023, 35000, 4.80, 9000, 26000, 62.00),
    -> ('Delhi', 2023, 42000, 5.60, 10000, 32000, 55.00),
    -> ('Bangalore', 2023, 29000, 4.10, 7500, 21500, 64.00),
    -> ('Kolkata', 2023, 27000, 3.70, 6000, 21000, 67.00),
    -> ('Chennai', 2023, 33000, 4.50, 8500, 24500, 60.00),
    -> ('Hyderabad', 2023, 31000, 4.20, 8000, 23000, 65.00),
    -> ('Pune', 2023, 25000, 3.60, 5500, 19500, 70.00),
    -> ('Ahmedabad', 2023, 28000, 4.00, 7000, 21000, 63.00),
    -> ('Jaipur', 2023, 22000, 3.80, 5000, 17000, 66.00),
    -> ('Chandigarh', 2023, 12000, 3.30, 2500, 9500, 72.00),
    -> ('Surat', 2023, 15000, 3.50, 3000, 12000, 68.00),
    -> ('Lucknow', 2023, 27000, 4.30, 6500, 20500, 61.00),
    -> ('Kochi', 2023, 19000, 3.00, 4500, 14500, 74.00),
    -> ('Indore', 2023, 15000, 3.20, 3500, 11500, 69.00),
    -> ('Nagpur', 2023, 17000, 3.50, 4000, 13000, 60.00),
    -> ('Bhopal', 2023, 14000, 3.00, 3000, 11000, 65.00),
    -> ('Visakhapatnam', 2023, 18000, 3.60, 4200, 13800, 62.00),
    -> ('Noida', 2023, 16000, 4.00, 4000, 12000, 64.00),
    -> ('Gurugram', 2023, 19000, 4.20, 4500, 14500, 66.00),
    -> ('Varanasi', 2023, 12000, 3.10, 2800, 9200, 70.00),
    -> ('Patna', 2023, 22000, 4.30, 5400, 16600, 58.00),
    -> ('Nagaland', 2023, 8000, 2.80, 2000, 6000, 75.00);
Query OK, 22 rows affected (0.01 sec)
Records: 22  Duplicates: 0  Warnings: 0
```



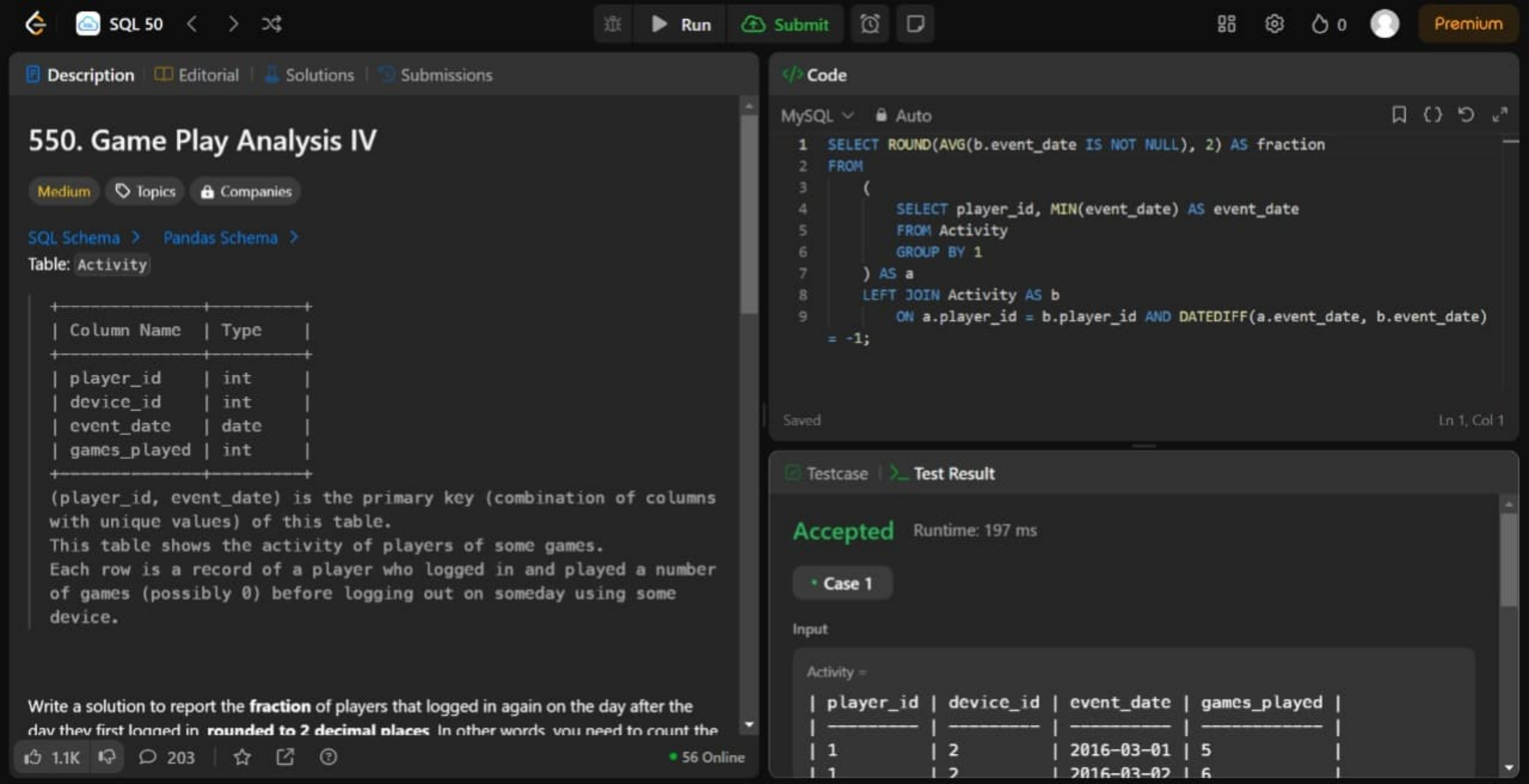
```
mysql> SELECT CityName, AVG(CrimeRate) AS AvgCrimeRate
    -> FROM CityCrimeStatistics
    -> GROUP BY CityName
    -> ORDER BY AvgCrimeRate DESC;
+---------------+--------------+
| CityName      | AvgCrimeRate |
+---------------+--------------+
| Delhi         |     5.600000 |
| Mumbai        |     4.800000 |
| Chennai       |     4.500000 |
| Lucknow       |     4.300000 |
| Patna         |     4.300000 |
| Hyderabad     |     4.200000 |
| Gurugram      |     4.200000 |
| Bangalore     |     4.100000 |
| Ahmedabad     |     4.000000 |
| Noida         |     4.000000 |
| Jaipur        |     3.800000 |
| Kolkata       |     3.700000 |
| Pune          |     3.600000 |
| Visakhapatnam |     3.600000 |
| Surat         |     3.500000 |
| Nagpur        |     3.500000 |
| Chandigarh    |     3.300000 |
| Indore        |     3.200000 |
| Varanasi      |     3.100000 |
| Kochi         |     3.000000 |
| Bhopal        |     3.000000 |
| Nagaland      |     2.800000 |
+---------------+--------------+
22 rows in set (0.00 sec)
```



```
mysql> SELECT CityName, SUM(ViolentCrimes) AS TotalViolentCrimes, SUM(PropertyCrimes) AS TotalPropertyCrimes
    -> FROM CityCrimeStatistics
    -> GROUP BY CityName
    -> ORDER BY TotalViolentCrimes DESC;
+---------------+--------------------+---------------------+
| CityName      | TotalViolentCrimes | TotalPropertyCrimes |
+---------------+--------------------+---------------------+
| Delhi         |              10000 |               32000 |
| Mumbai        |               9000 |               26000 |
| Chennai       |               8500 |               24500 |
| Hyderabad     |               8000 |               23000 |
| Bangalore     |               7500 |               21500 |
| Ahmedabad     |               7000 |               21000 |
| Lucknow       |               6500 |               20500 |
| Kolkata       |               6000 |               21000 |
| Pune          |               5500 |               19500 |
| Patna         |               5400 |               16600 |
| Jaipur        |               5000 |               17000 |
| Kochi         |               4500 |               14500 |
| Gurugram      |               4500 |               14500 |
| Visakhapatnam |               4200 |               13800 |
| Nagpur        |               4000 |               13000 |
| Noida         |               4000 |               12000 |
| Indore        |               3500 |               11500 |
| Surat         |               3000 |               12000 |
| Bhopal        |               3000 |               11000 |
| Varanasi      |               2800 |                9200 |
| Chandigarh    |               2500 |                9500 |
| Nagaland      |               2000 |                6000 |
+---------------+--------------------+---------------------+
22 rows in set (0.00 sec)
```



```
mysql> SELECT CityName, AVG(ClearanceRate) AS AvgClearanceRate
    -> FROM CityCrimeStatistics
    -> GROUP BY CityName
    -> ORDER BY AvgClearanceRate DESC;
+---------------+------------------+
| CityName      | AvgClearanceRate |
+---------------+------------------+
| Nagaland      |        75.000000 |
| Kochi         |        74.000000 |
| Chandigarh    |        72.000000 |
| Pune          |        70.000000 |
| Varanasi      |        70.000000 |
| Indore        |        69.000000 |
| Surat         |        68.000000 |
| Kolkata       |        67.000000 |
| Jaipur        |        66.000000 |
| Gurugram      |        66.000000 |
| Hyderabad     |        65.000000 |
| Bhopal        |        65.000000 |
| Bangalore     |        64.000000 |
| Noida         |        64.000000 |
| Ahmedabad     |        63.000000 |
| Mumbai        |        62.000000 |
| Visakhapatnam |        62.000000 |
| Lucknow       |        61.000000 |
| Chennai       |        60.000000 |
| Nagpur        |        60.000000 |
| Patna         |        58.000000 |
| Delhi         |        55.000000 |
+---------------+------------------+
22 rows in set (0.00 sec)

mysql> SELECT Year, AVG(CrimeRate) AS AvgCrimeRate
    -> FROM CityCrimeStatistics
    -> GROUP BY Year
    -> ORDER BY Year;
+------+--------------+
| Year | AvgCrimeRate |
+------+--------------+
| 2023 |     3.822727 |
+------+--------------+
1 row in set (0.00 sec)
```



550. Game Play Analysis IV

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: `Activity`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
(player_id, event_date) is the primary key (combination of columns
with unique values) of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number
of games (possibly 0) before logging out on someday using some
device.
```

Write a solution 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

```sql
SELECT ROUND(AVG(b.event_date IS NOT NULL), 2) AS fraction
FROM
    (
        SELECT player_id, MIN(event_date) AS event_date
        FROM Activity
        GROUP BY 1
    ) AS a
    LEFT JOIN Activity AS b
        ON a.player_id = b.player_id AND DATEDIFF(a.event_date, b.event_date)
= -1;
```

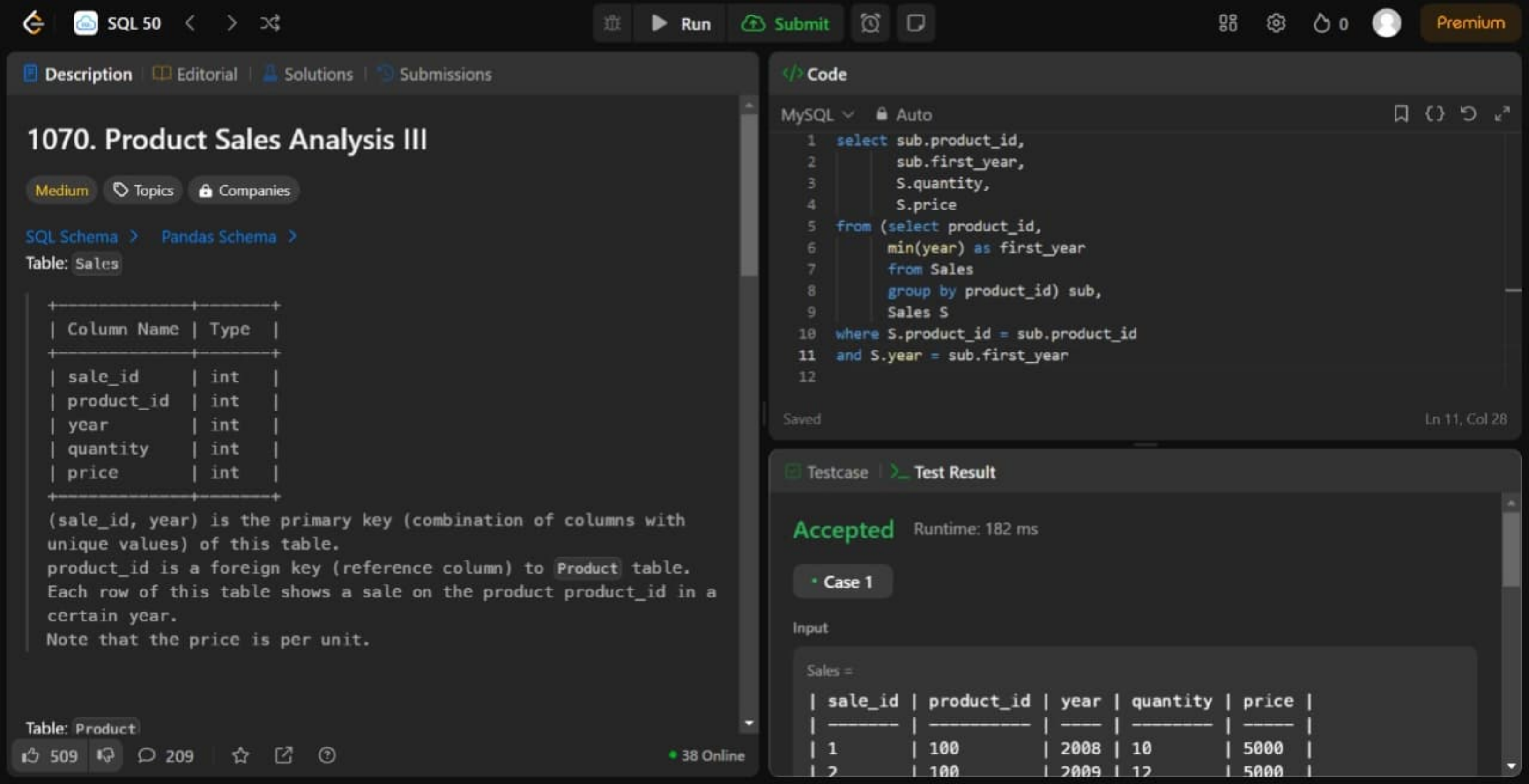
Accepted Runtime: 197 ms

Case 1

Input

Activity =

```
| player_id | device_id | event_date | games_played |
| --------- | --------- | ---------- | ------------ |
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-03-02 | 6            |
```



SQL 50 Run Submit Premium

Description Editorial Solutions Submissions

1070. Product Sales Analysis III

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: `Sales`

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
(sale_id, year) is the primary key (combination of columns with
unique values) of this table.
product_id is a foreign key (reference column) to Product table.
Each row of this table shows a sale on the product product_id in a
certain year.
Note that the price is per unit.
```

Table: `Product`

509 209 38 Online

Code

MySQL Auto

```sql
select sub.product_id,
       sub.first_year,
       S.quantity,
       S.price
from (select product_id,
      min(year) as first_year
      from Sales
      group by product_id) sub,
      Sales S
where S.product_id = sub.product_id
and S.year = sub.first_year
```

Saved Ln 11, Col 28

Testcase Test Result

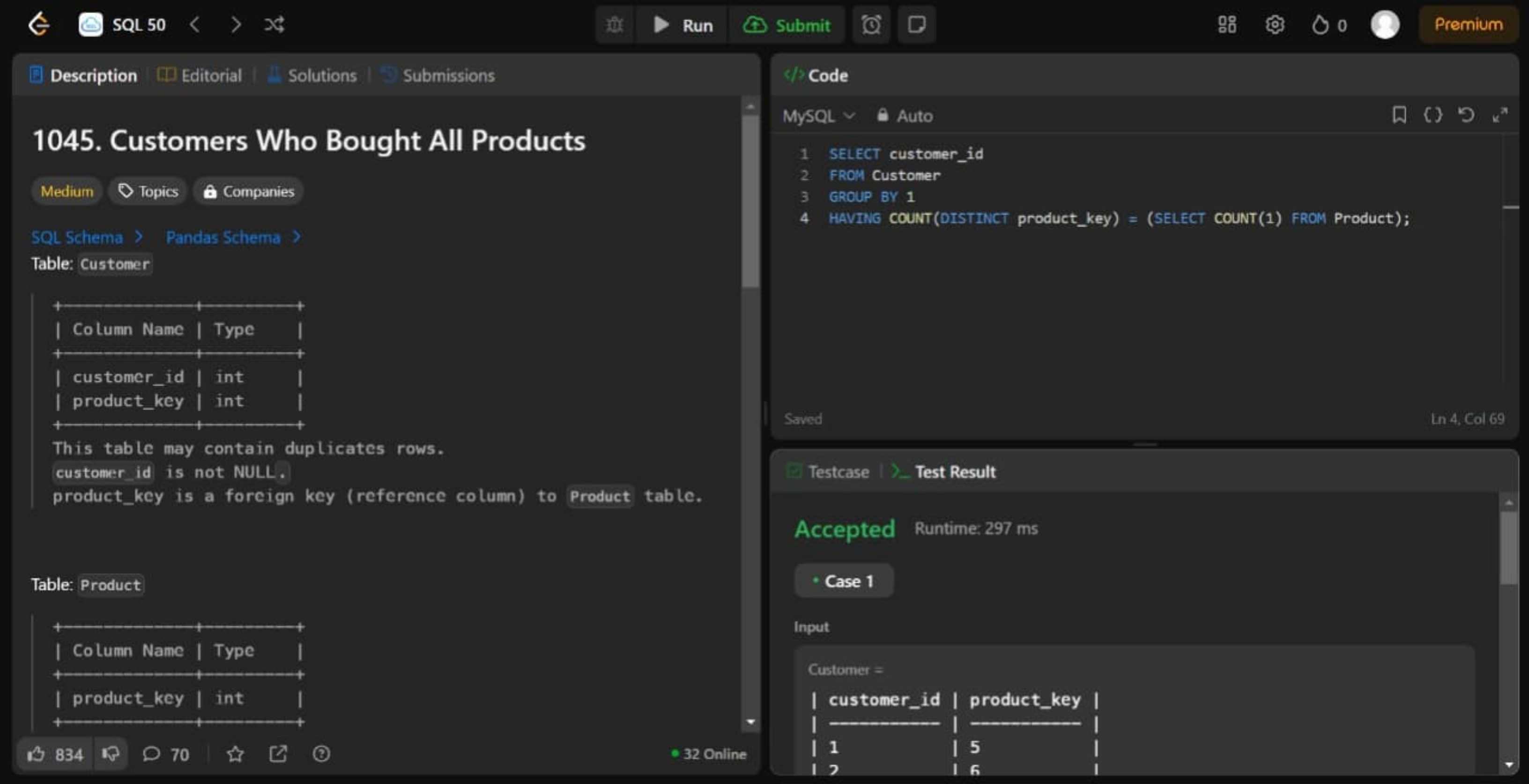
Accepted Runtime: 182 ms

Case 1

Input

Sales =

```
| sale_id | product_id | year | quantity | price |
| ------- | ---------- | ---- | -------- | ----- |
| 1       | 100        | 2008 | 10       | 5000  |
| 2       | 100        | 2009 | 12       | 5000  |
```



1045. Customers Who Bought All Products

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: `Customer`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| customer_id | int     |
| product_key | int     |
+-------------+---------+
This table may contain duplicates rows.
customer_id is not NULL.
product_key is a foreign key (reference column) to Product table.
```

Table: `Product`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_key | int     |
+-------------+---------+
```

834 70 32 Online

```sql
SELECT customer_id
FROM Customer
GROUP BY 1
HAVING COUNT(DISTINCT product_key) = (SELECT COUNT(1) FROM Product);
```

Saved Ln 4, Col 69

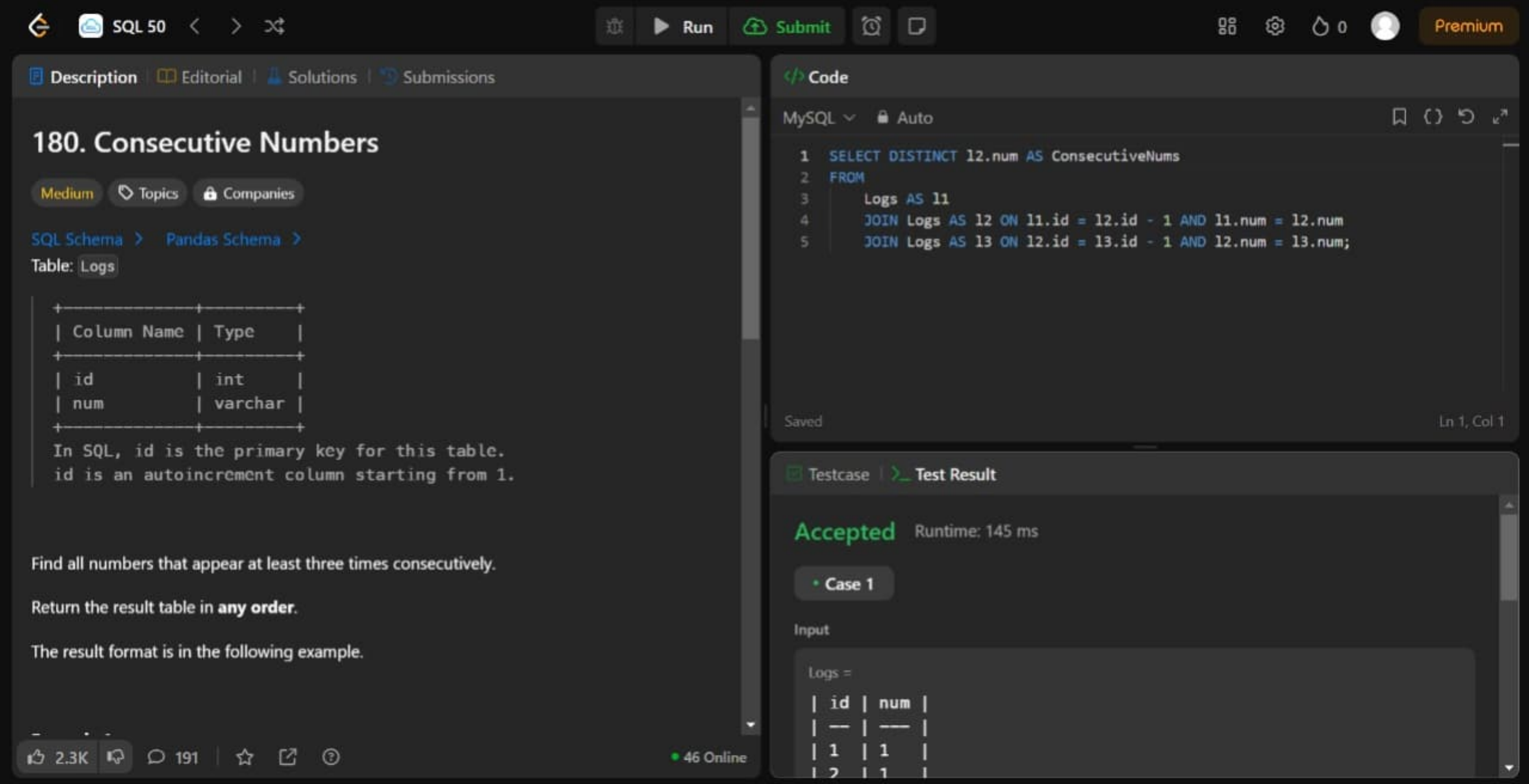
Accepted Runtime: 297 ms

Case 1

Input

Customer =

```
| customer_id | product_key |
| ----------- | ----------- |
| 1           | 5           |
| 2           | 6           |
```



180. Consecutive Numbers

SQL Schema > Pandas Schema >

Table: Logs

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| num         | varchar |
+-------------+---------+
In SQL, id is the primary key for this table.
id is an autoincrement column starting from 1.
```

Find all numbers that appear at least three times consecutively.

Return the result table in **any order**.

The result format is in the following example.

```sql
SELECT DISTINCT l2.num AS ConsecutiveNums
FROM
    Logs AS l1
    JOIN Logs AS l2 ON l1.id = l2.id - 1 AND l1.num = l2.num
    JOIN Logs AS l3 ON l2.id = l3.id - 1 AND l2.num = l3.num;
```

Accepted Runtime: 145 ms

Case 1

Input

Logs =

```
| id | num |
| -- | --- |
| 1  | 1   |
| 2  | 1   |
```