HELLENIC REPUBLIC
National and Kapodistrian University of Athens
EST. 1837

DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS

Programming Project I
311 Chicago Incidents System

Fotis Memis - cs2200010
Anna-Aikaterini Kavvada - 1115201500050

December 7, 2020

Professor: Alexis Delis
Class: M149 - Fall 2020

1 Introduction

In this project we have designed and implemented a database solution to manage *"311 Incidents"* data openly published by the city of Chicago, IL. Along with it we have also designed a web application, providing access to the aforementioned database to the authorised users, namely the residents of Chicago. The database termed $db_311ciispopulatedbydataavailableat :$ $https : //www.kaggle.com/chicago/chicago - 311 - service - requests.$

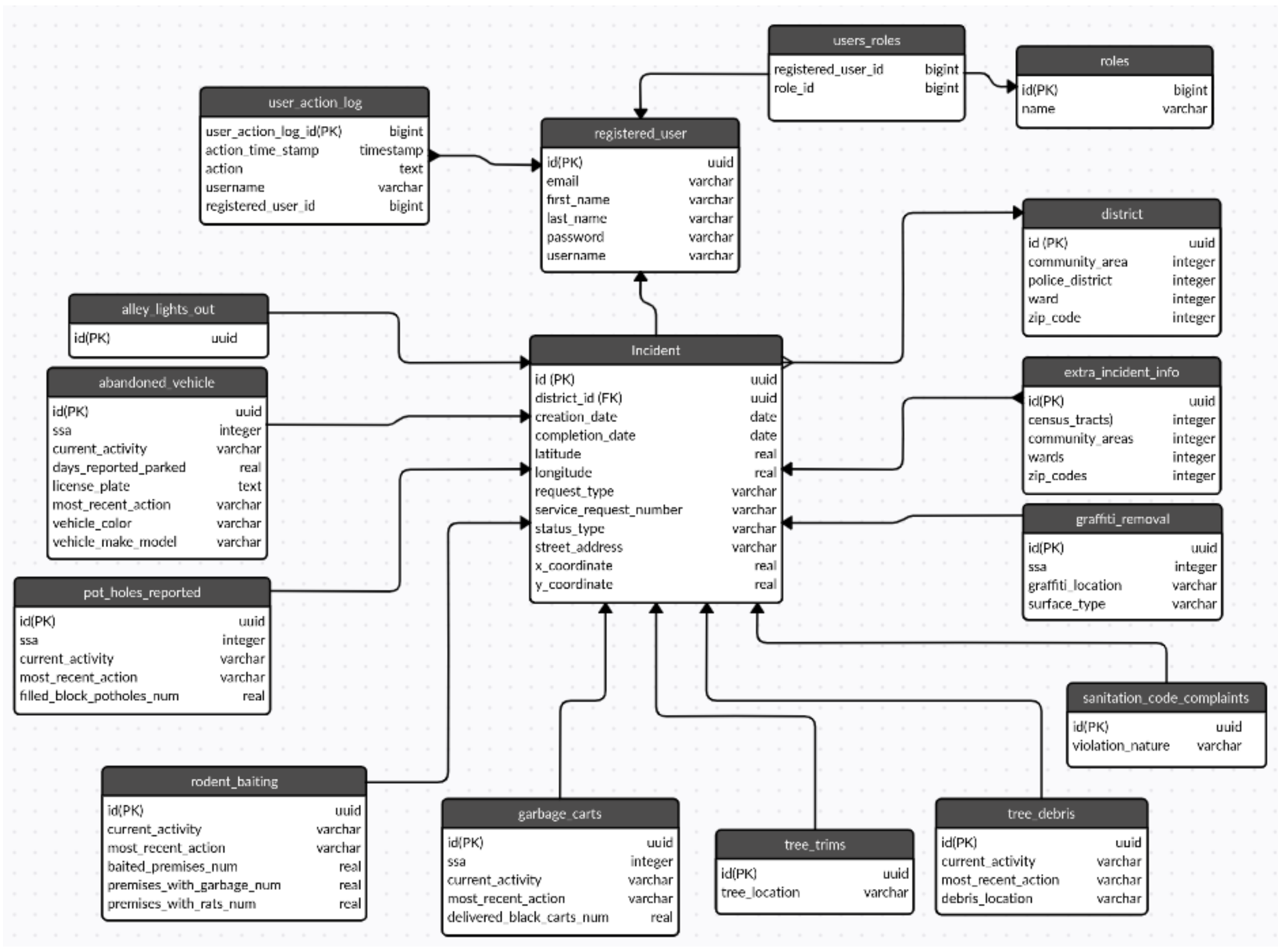
2 Project Structure

- ***project1***
  - ***build***
  - ***gradle***
  - ***src***
    - ***main***
      - ***java:*** *the back-end implementation*
      - ***resources:*** *the front-end design implementation*
    - ***test***
- ***SQLs***
  - ***exampleQueries.sql:*** *all the native queries implemented for the application*
  - ***import_CSVs_to_db_table.py:*** *python script used to import the CSVs in database tables (migration phase 1)*
  - ***indexes.sql:*** *native sql for the indexes we used in our database solution*
  - ***migration.sql:*** *insertion queries to insert the data in the normalised data base schema we have designed (migration phase 2)*
- ***Report***

3 Data Base Schema, Indices and Data Migration

3.1 Schema

After analyzing the ”311 Incidents” data and the queries that our database solution had to execute, we came up with the database schema below.



The database solution we provide, consists of 17 table. That is 5 tables for the system's registered users and the user log data and 13 tables for the reported incidents data. The idea is, that we have a main table called incident and an auxiliary table called extra_incident_info, 9 tables, one for each specific incident type, except for the cases of the incidents referring to light outage (Lights All Out, Street Light One Out, Alley Lights Out). These latter incidents, we considered them of the same "sub-incident type". Last we have the district table. The incident table stores all the common data among all the types of incidents. The extra_incident_info table is an "extension" of the incident table, having a 1-1 type of relationship and containing some additional information for each incident which in many cases is omitted. Each sub-incident table stores the data that define this specific category of incidents (e.g. we will not find an incident report for graffiti removal having a license plate number). The district table contains unique quadruples of (zip_code, police_district, ward, community_area), defining a specific and unique Chicago service district. The choice for this table was made, after the observation that in the 4 million entries of the database there are only 1300 different combinations of these four particular attributes.

3.2 Indices

To ensure the uniqueness of the registrations in table district, we added an index witch ensures this constraint among the quadruples that each table row stores.
In order to optimize the execution time of our database transactions (insert, update, search) we added 2 indices. The differences can be observed in the metrics section of the report right below.
The first index we added is between the ids of the incident table and the request types that exist in the database. Thus we map, in a way, the sections of the table according to the report type they have.
he second index is on request_service_number. With this index, when we need to make a search query with the request_service_number, we observed that the execution time drops by 90% in comparison with the search without the index. Furthermore, in this kind of applications it is expected that the mass amount of transactions will be based on this particular number.

3.3 Data Migration

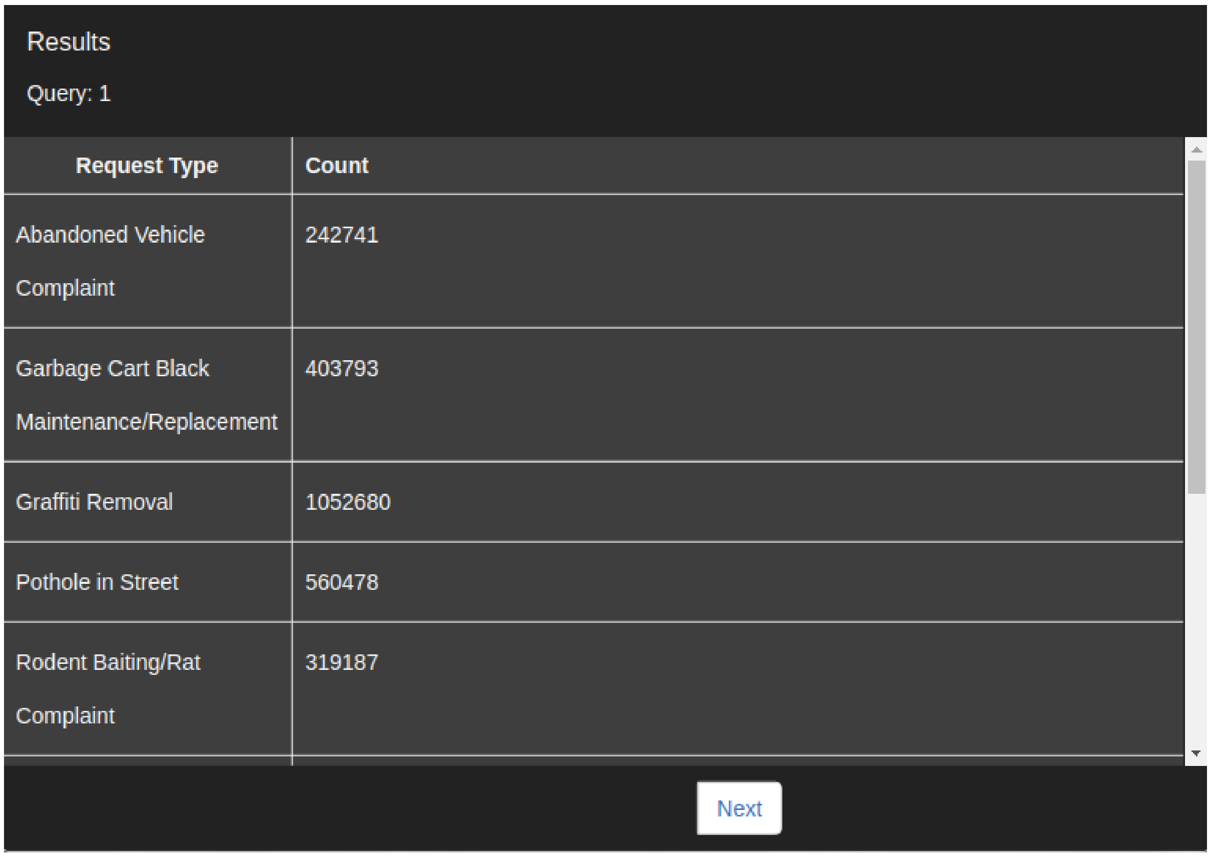
To populate the database with the dataset of all the incidents we were given in the form of csv files, we used the tool pgfutter. We implemented a python script, that uses the pgfutter to import in auxiliary tables in schema import of the database all the csv files. Following that, we applied insertion queries in order to distribute the data in our own schema. Finally, we deleted all the auxiliary tables created from the pgfutter. It should be noted that our migration is completed only in 6 minutes.

4 Web Application

For the web application we used the following technologies:
Spring boot with spring security was used in order to implement the app. To communicate with the database, we took advantage of the JPA repository interface(implemented internally by hibernate) which allowed us to limit extensively the boilerplate code for the queries. We also included lombok plugin which makes use of annotations in order to automatically create repetitive code such as getters, setters and constructors, behind the scenes. User authentication and access rights to specific pages, were implemented with Spring Security. The front end was implemented using html, css and javascript, along with some bootstrap and templates we found online.

5 Sample snapshots of the query results - Metrics

1. Find the total requests per type that were created within a specified time range and sort them in a descending order.

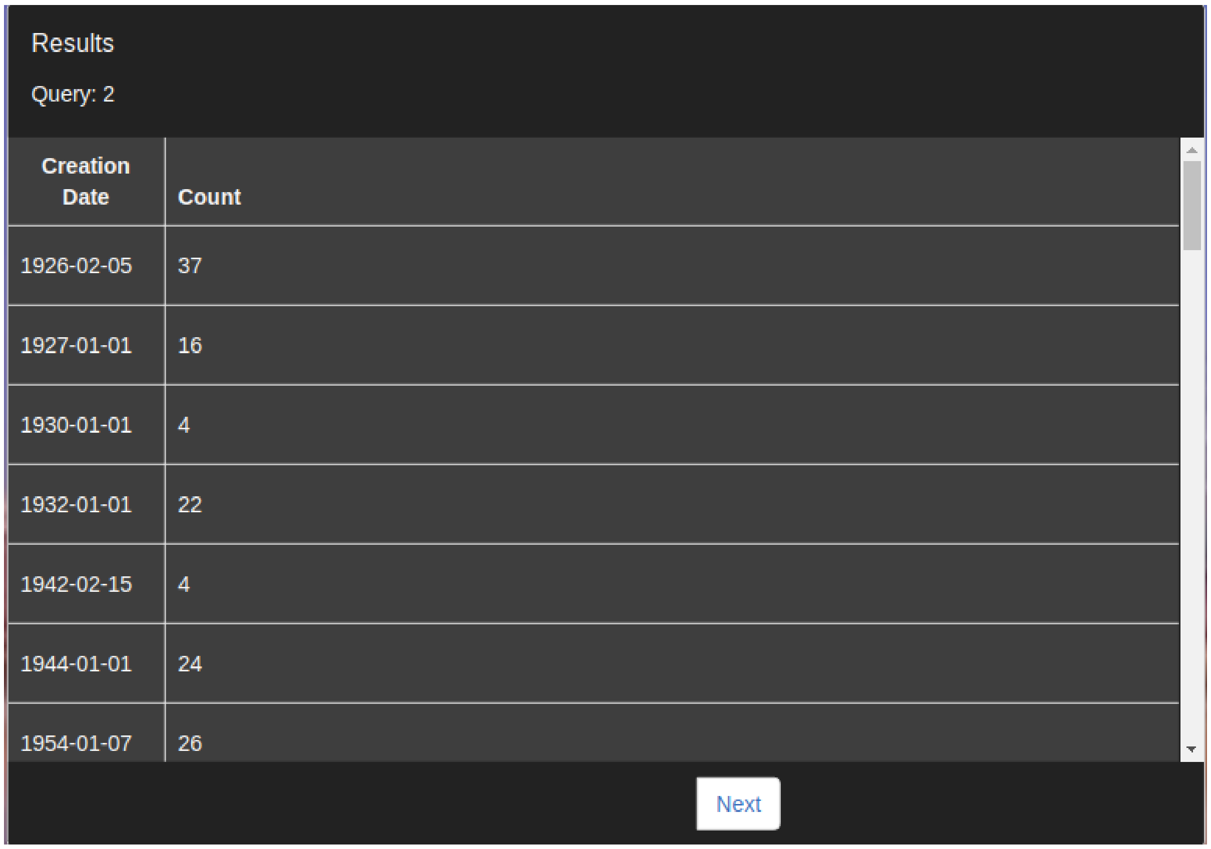


Results

Query: 1

Request Type	Count
Abandoned Vehicle Complaint	242741
Garbage Cart Black Maintenance/Replacement	403793
Graffiti Removal	1052680
Pothole in Street	560478
Rodent Baiting/Rat Complaint	319187

2. Find the total requests per day for a specific request type and time range.

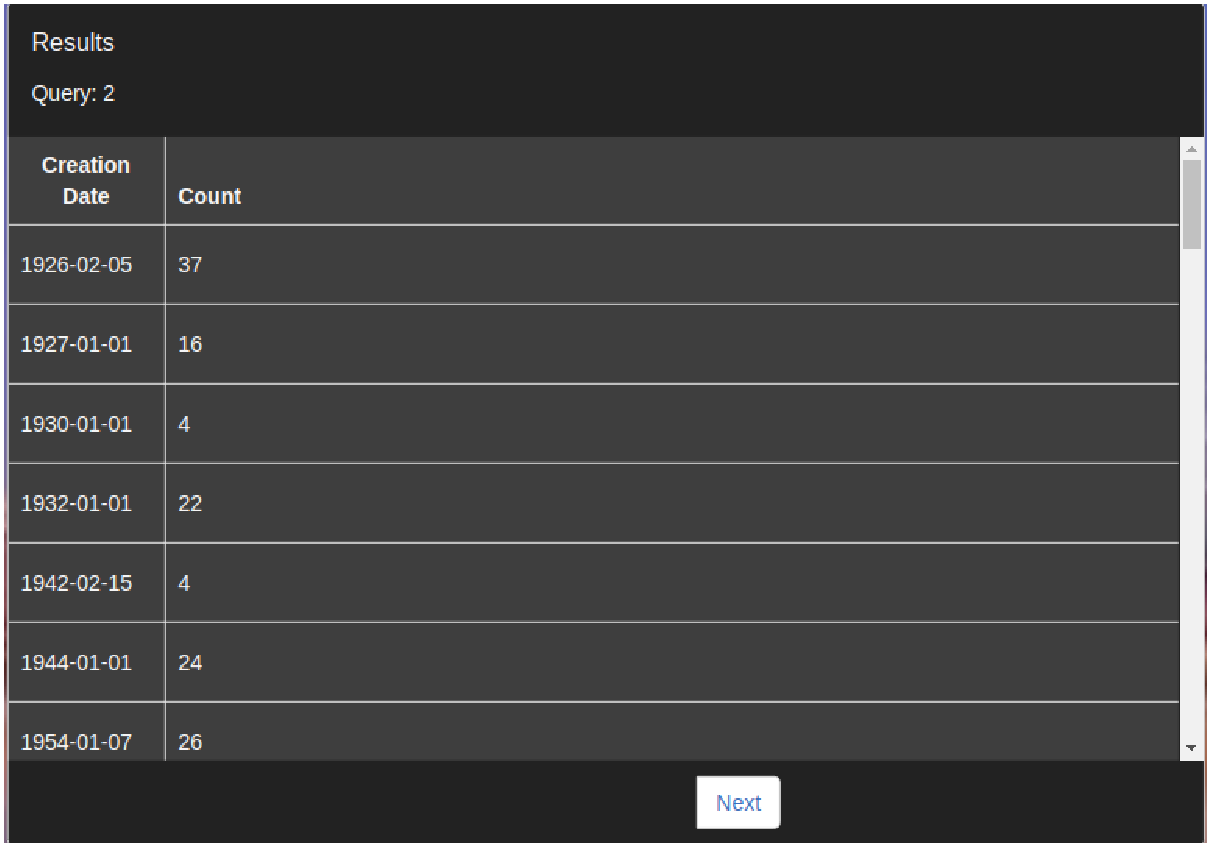


Results

Query: 2

Creation Date	Count
1926-02-05	37
1927-01-01	16
1930-01-01	4
1932-01-01	22
1942-02-15	4
1944-01-01	24
1954-01-07	26

3. Find the most common service request per zipcode for a specific day.

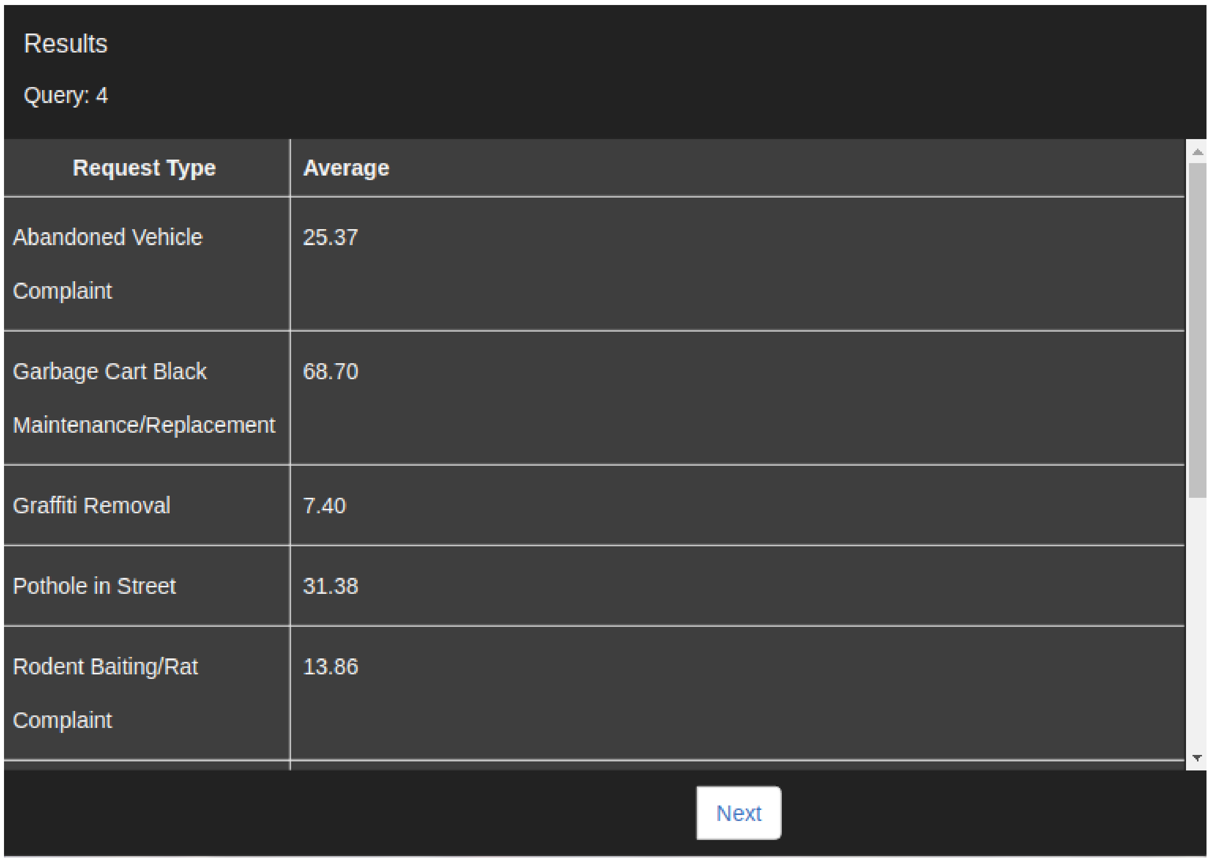


Results

Query: 2

Creation Date	Count
1926-02-05	37
1927-01-01	16
1930-01-01	4
1932-01-01	22
1942-02-15	4
1944-01-01	24
1954-01-07	26

4. Find the average completion time per service request for a specific date range.

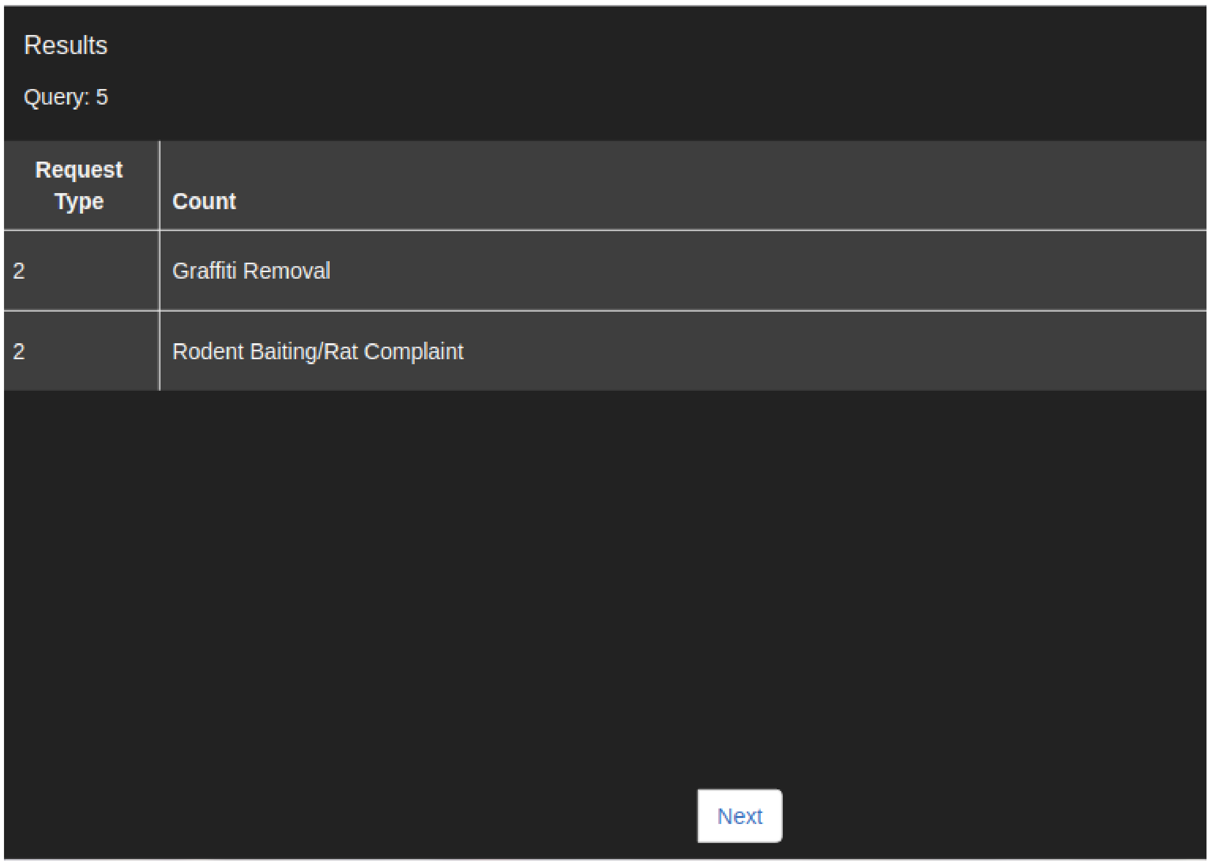


Results

Query: 4

Request Type	Average
Abandoned Vehicle Complaint	25.37
Garbage Cart Black Maintenance/Replacement	68.70
Graffiti Removal	7.40
Pothole in Street	31.38
Rodent Baiting/Rat Complaint	13.86

5. Find the most common service request in a specified bounding box (as designated by GPScoordinates) for a specific day.

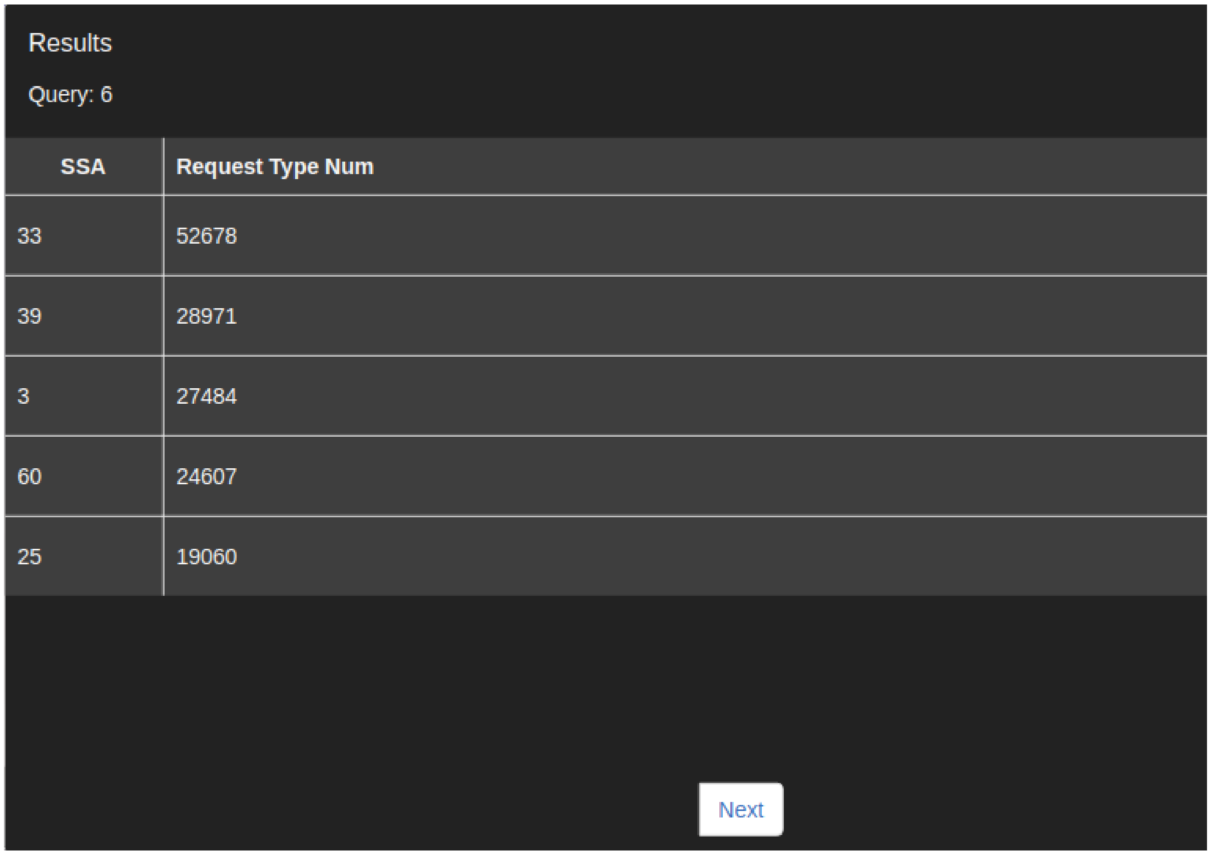


Results

Query: 5

Request Type	Count
2	Graffiti Removal
2	Rodent Baiting/Rat Complaint

6. Find the top-5 Special Service Areas (SSA) with regards to total number of requests per day for a specific date range (for service requests types that SSA is available: abandoned vehicles, garbage carts, graffiti removal, pot holes reported)

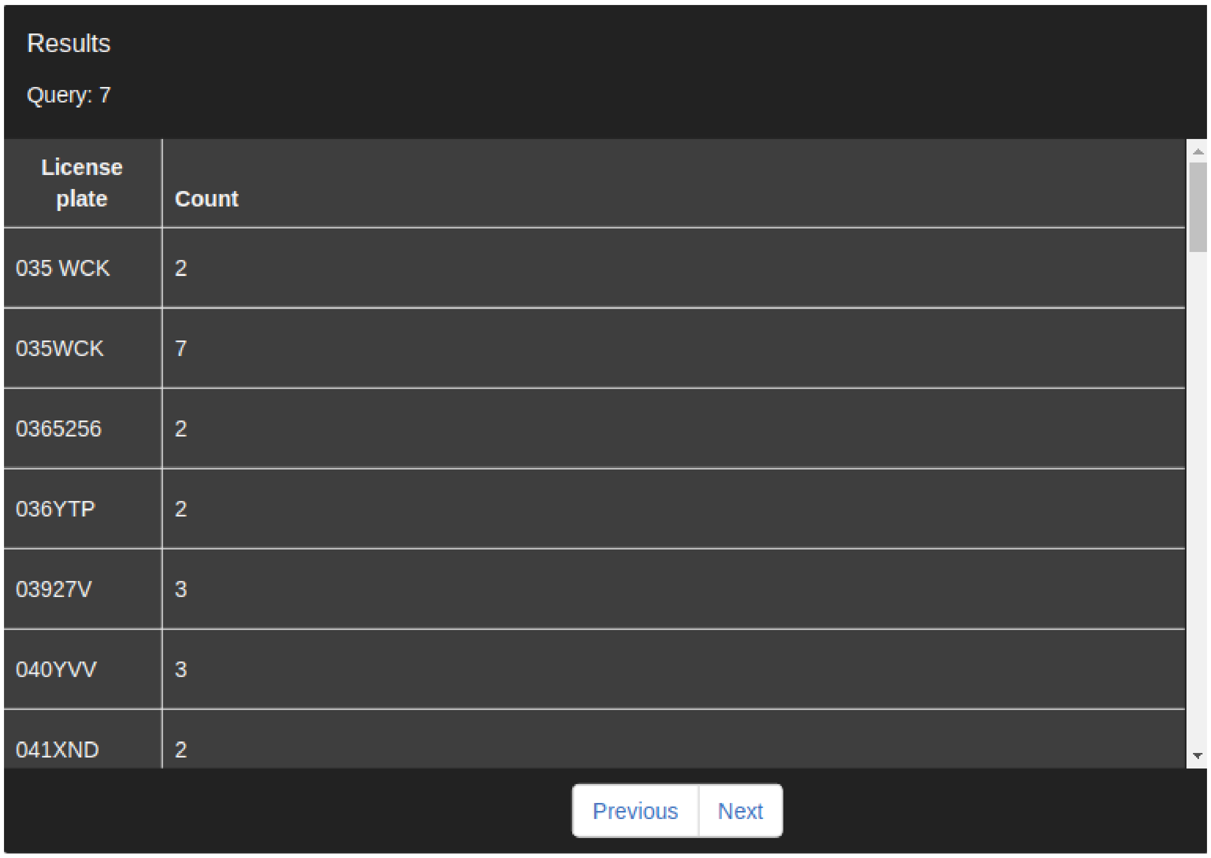


Results

Query: 6

SSA	Request Type Num
33	52678
39	28971
3	27484
60	24607
25	19060

7. Find the license plates (if any) that have been involved in abandoned vehicle complaints more than once.

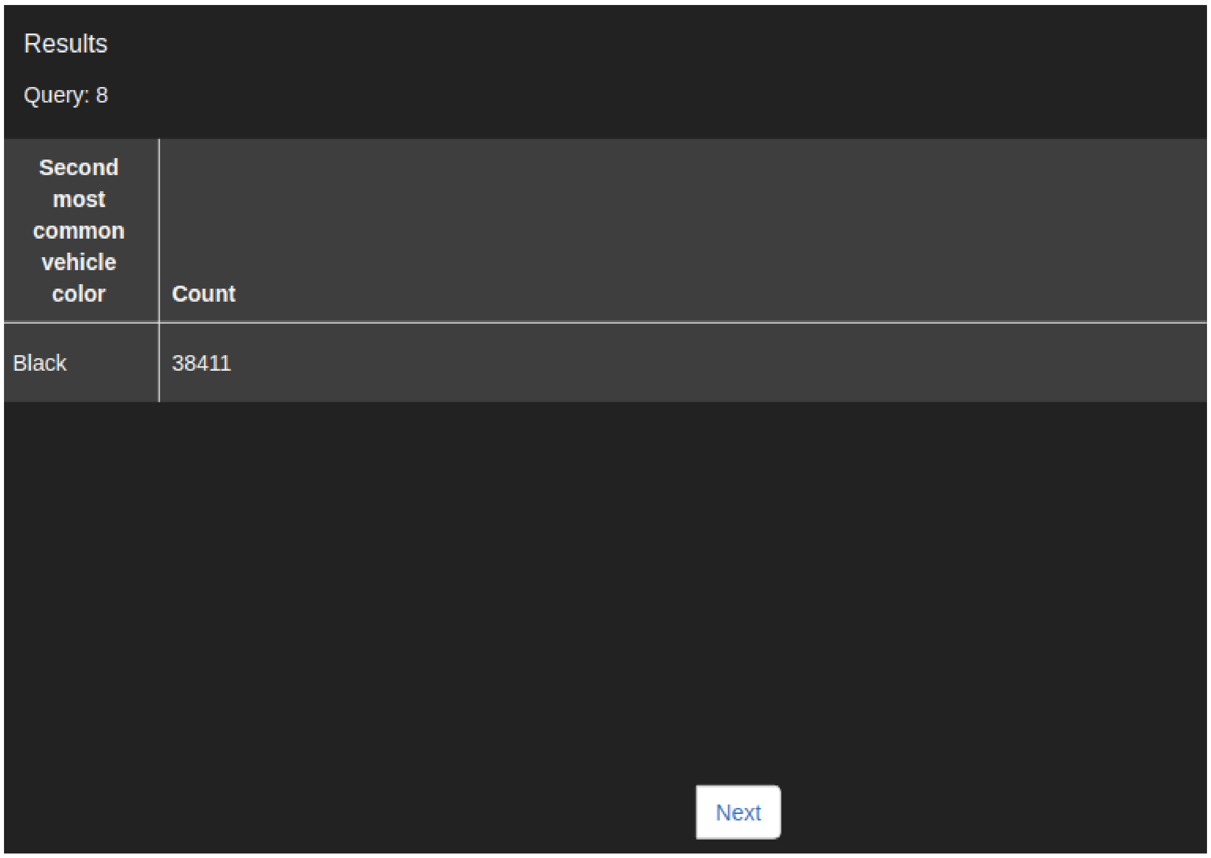


Results

Query: 7

License plate	Count
035 WCK	2
035WCK	7
0365256	2
036YTP	2
03927V	3
040YVV	3
041XND	2

8. Find the second most common color of vehicles involved in abandoned vehicle complaints.

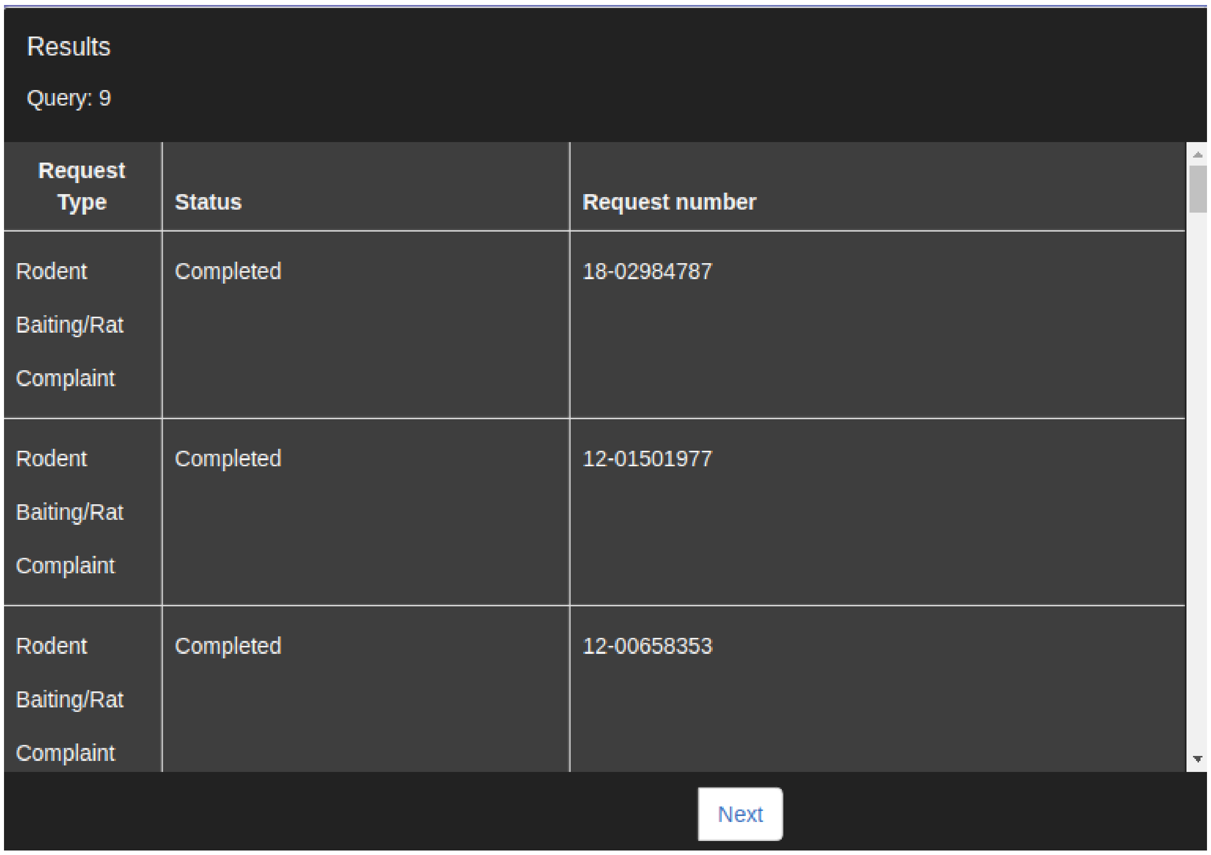


Results

Query: 8

Second most common vehicle color	Count
Black	38411

9. Find the rodent baiting requests where the number of premises baited is less than a specified number.

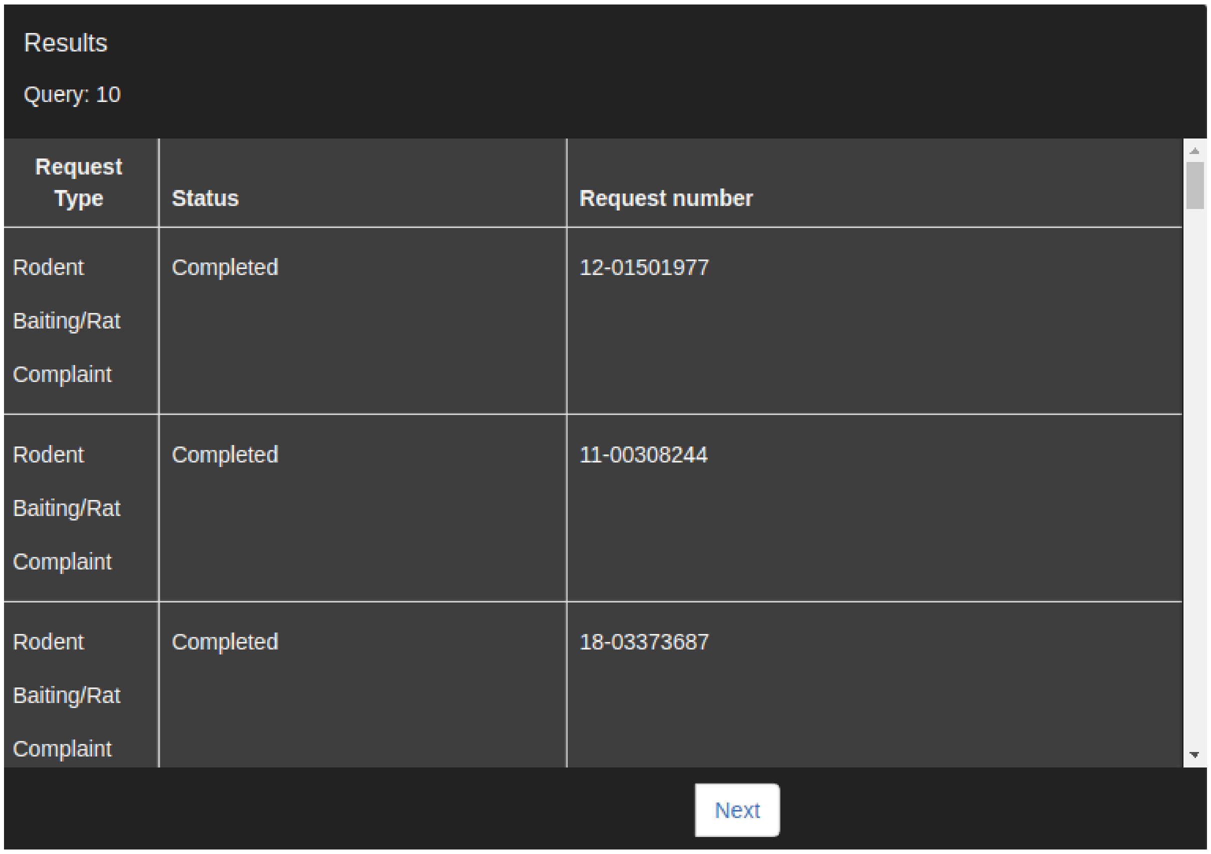


Results

Query: 9

Request Type	Status	Request number
Rodent Baiting/Rat Complaint	Completed	18-02984787
Rodent Baiting/Rat Complaint	Completed	12-01501977
Rodent Baiting/Rat Complaint	Completed	12-00658353

10. Same as the above (i.e., 9) for premises with garbage.

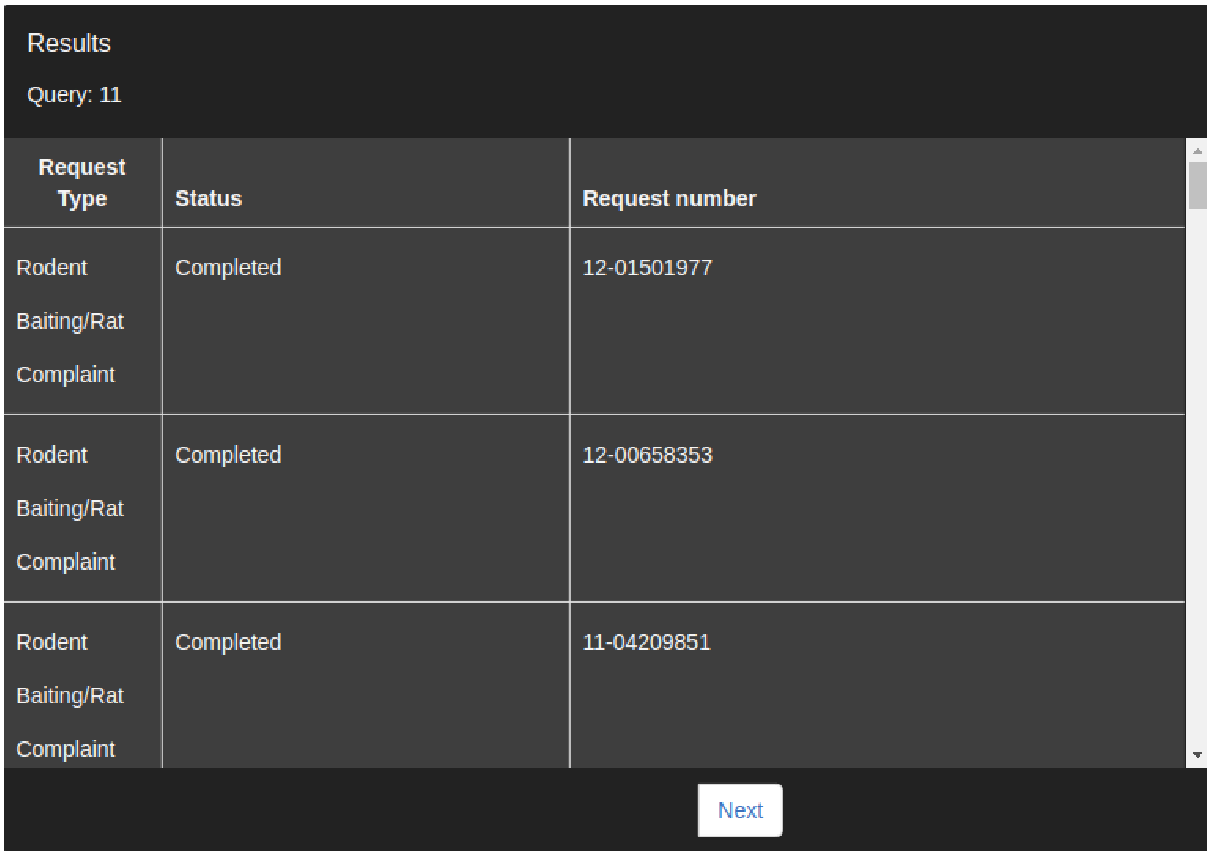


Results

Query: 10

Request Type	Status	Request number
Rodent Baiting/Rat Complaint	Completed	12-01501977
Rodent Baiting/Rat Complaint	Completed	11-00308244
Rodent Baiting/Rat Complaint	Completed	18-03373687

11. Same as the above (i.e., 10) for premises with rats.

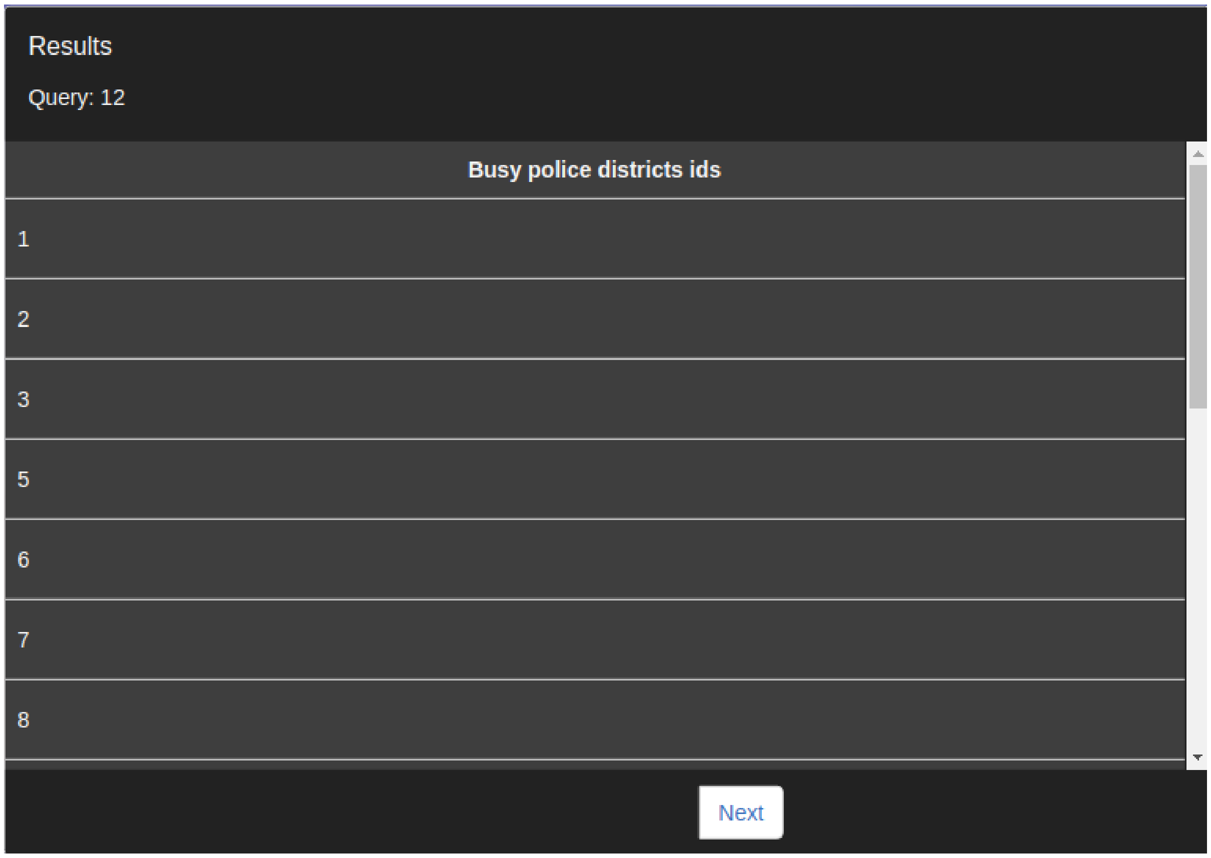


Results

Query: 11

Request Type	Status	Request number
Rodent Baiting/Rat Complaint	Completed	12-01501977
Rodent Baiting/Rat Complaint	Completed	12-00658353
Rodent Baiting/Rat Complaint	Completed	11-04209851

12. Find the police districts that have handled "pot holes" requests with more than one number of potholes on the same day that they also handled "rodent baiting" requests with more than one number of premises baited, for a specific day.

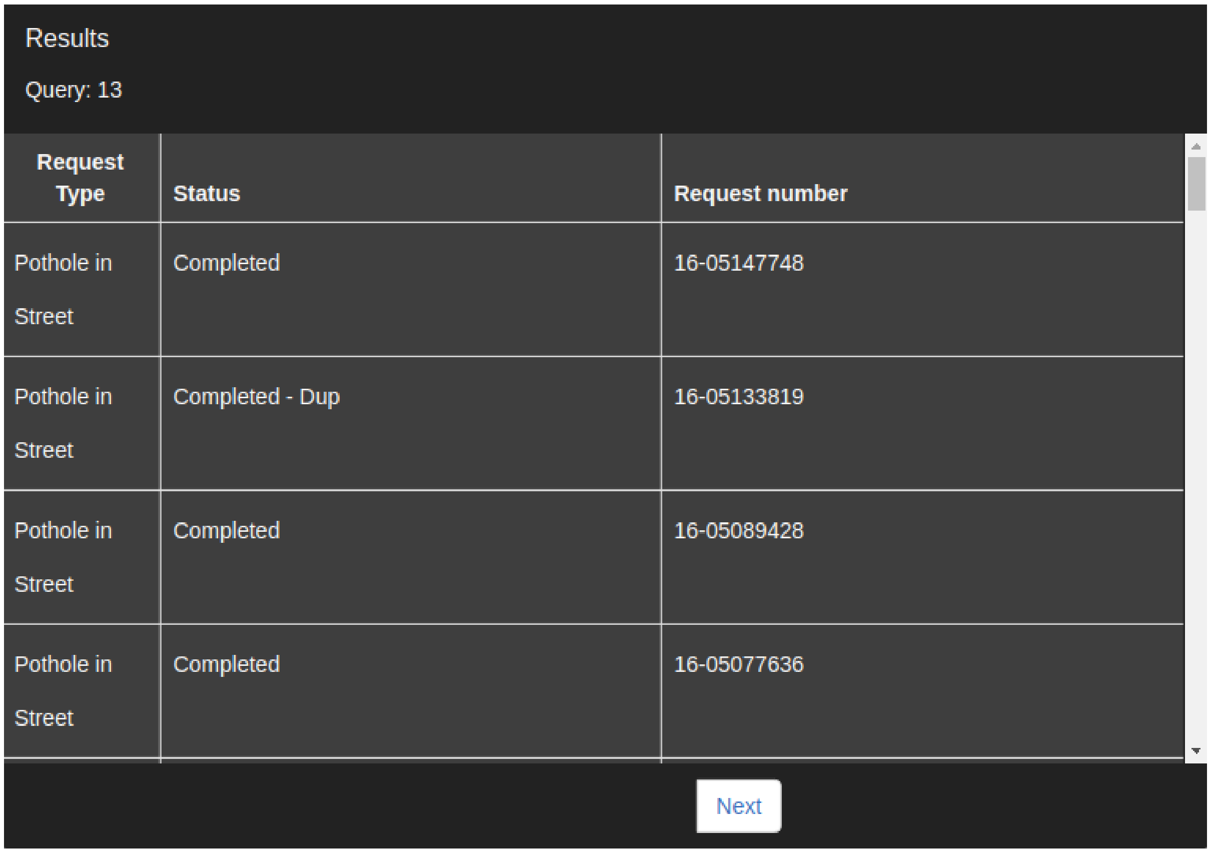


Results

Query: 12

Busy police districts ids
1
2
3
5
6
7
8

13. Find Incidents by zip code.

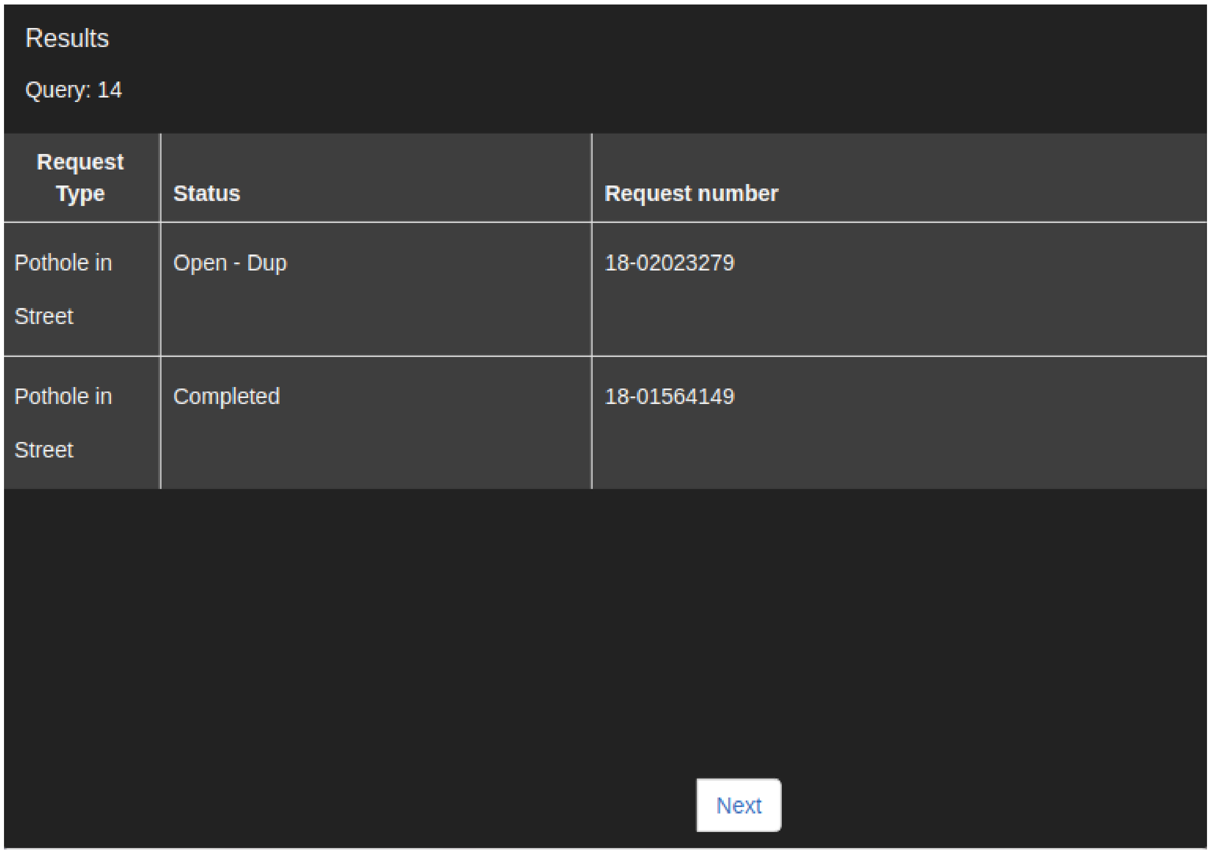


Results

Query: 13

Request Type	Status	Request number
Pothole in Street	Completed	16-05147748
Pothole in Street	Completed - Dup	16-05133819
Pothole in Street	Completed	16-05089428
Pothole in Street	Completed	16-05077636

14. Find Incidents by street address.

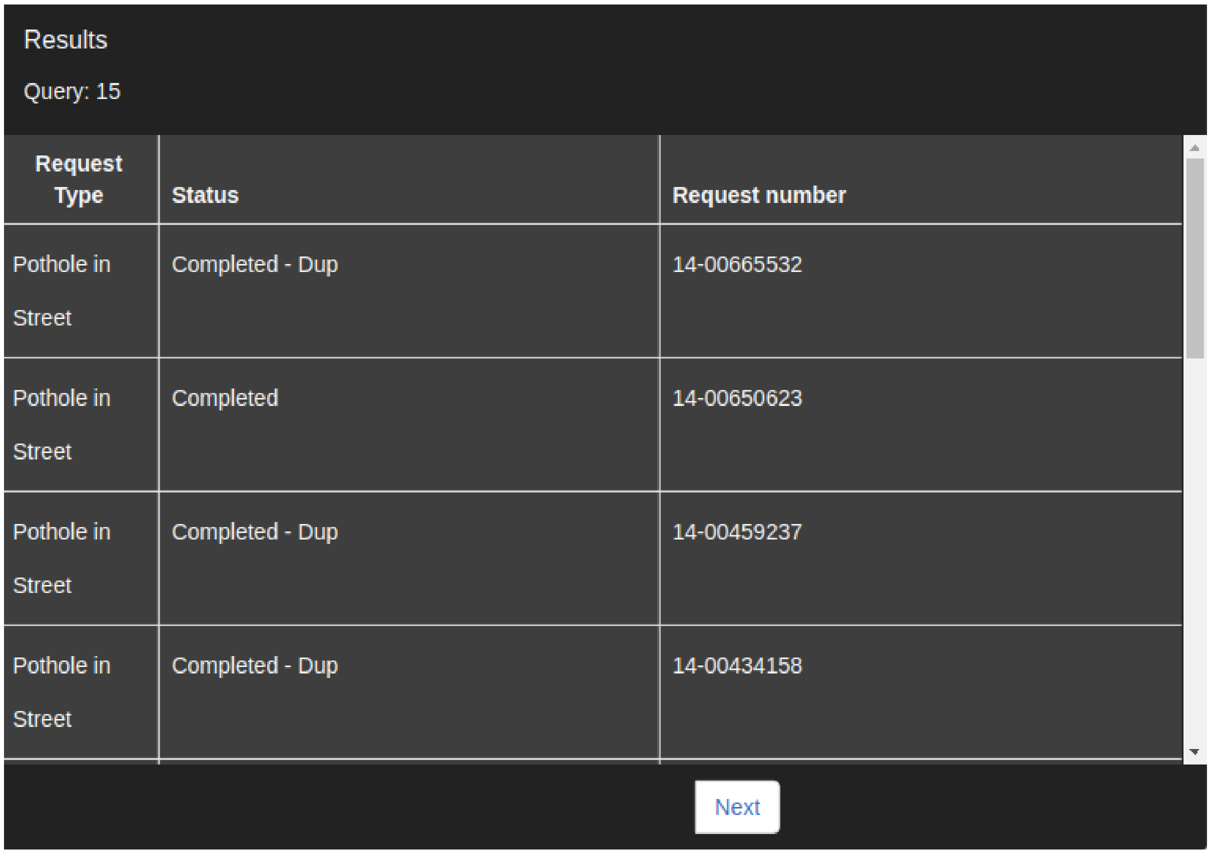


Results

Query: 14

Request Type	Status	Request number
Pothole in Street	Open - Dup	18-02023279
Pothole in Street	Completed	18-01564149

15. Find incidents by street address and zip code.



Results

Query: 15

Request Type	Status	Request number
Pothole in Street	Completed - Dup	14-00665532
Pothole in Street	Completed	14-00650623
Pothole in Street	Completed - Dup	14-00459237
Pothole in Street	Completed - Dup	14-00434158

6 Query Metrics

In this section we present you with snapshots of each query's execution time and results. All the execution times were measured for a time range of a century (from 1920 to 2020)

Measurements from the console, without the applied index.

- *Execution time of query 1 was 1 seconds and 125 milliseconds*
- *Execution time of query 2 was 0 seconds and 643 milliseconds*
- *Execution time of query 3 was 0 seconds and 462 milliseconds*
- *Execution time of query 4 was 0 seconds and 603 milliseconds*
- *Execution time of query 5 was 0 seconds and 933 milliseconds*
- *Execution time of query 6 was 17 seconds and 878 milliseconds*
- *Execution time of query 7 was 1 seconds and 205 milliseconds*
- *Execution time of query 8 was 0 seconds and 99 milliseconds*
- *Execution time of query 9 was 0 seconds and 690 milliseconds*
- *Execution time of query 10 was 0 seconds and 269 milliseconds*
- *Execution time of query 11 was 0 seconds and 135 milliseconds*
- *Execution time of query 12 was 13 seconds and 642 milliseconds*

Measurements from the query console, with the applied indices.

- *Execution time of query 1 was 0 seconds and 970 milliseconds*
- *Execution time of query 2 was 0 seconds and 125 milliseconds*
- *Execution time of query 3 was 0 seconds and 397 milliseconds*
- *Execution time of query 4 was 0 seconds and 232 milliseconds*
- *Execution time of query 5 was 0 seconds and 480 milliseconds*
- *Execution time of query 6 was 3 seconds and 711 milliseconds*
- *Execution time of query 7 was 0 seconds and 510 milliseconds*
- *Execution time of query 8 was 0 seconds and 68 milliseconds*
- *Execution time of query 9 was 0 seconds and 127 milliseconds*
- *Execution time of query 10 was 0 seconds and 92 milliseconds*
- *Execution time of query 11 was 0 seconds and 95 milliseconds*
- *Execution time of query 12 was 0 seconds and 808 milliseconds*