11/18/2018

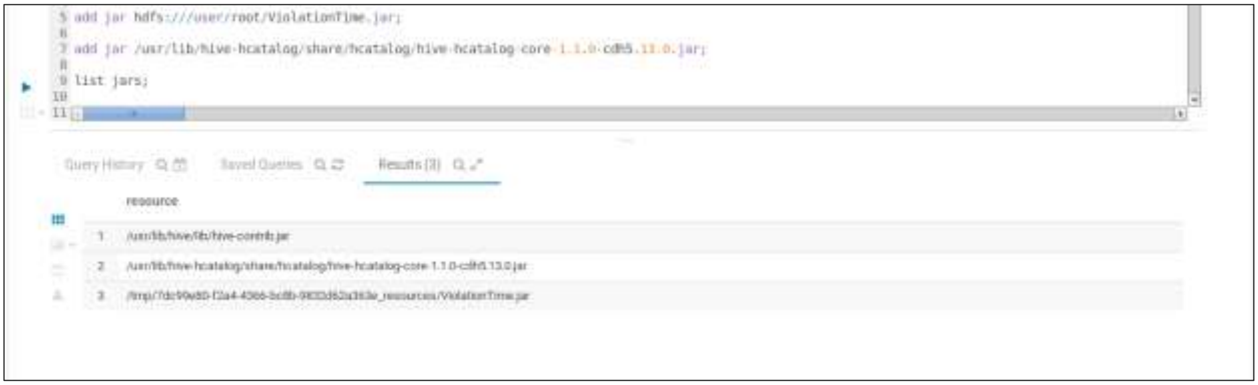
Examine & Analyze the New York City Parking Violations Data

Hive Assignment

Name:Ruksana Bhanu

JARS ADDED:

- add jar hdfs:///user/root/ViolationTime.jar;
- add jar /usr/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.1.0-cdh5.13.0.ja

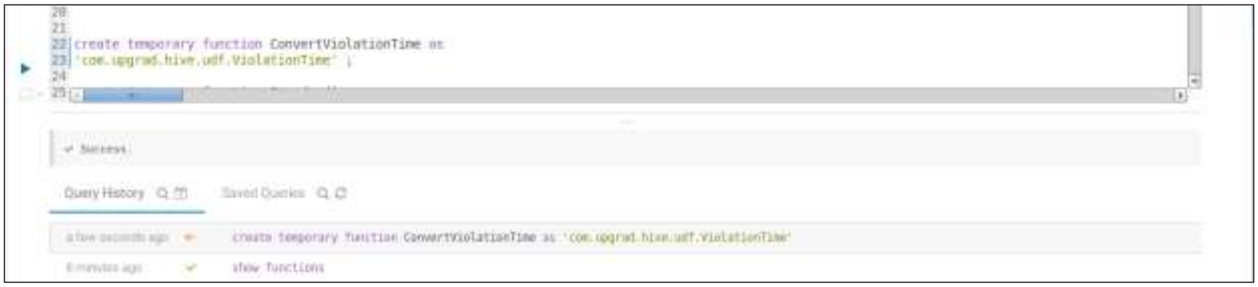


```
add jar hdfs:///user/root/ViolationTime.jar;

add jar /usr/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.1.0-cdh5.13.0.jar;

list jars;
```

	resource
1	/usr/lib/hive/lib/hive-contrib.jar
2	/usr/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.1.0-cdh5.13.0.jar
3	/tmp/7dc99e80-f2a4-4366-bc8b-9833d62a363e_resources/ViolationTime.jar

FUNCTIONS CREATED:

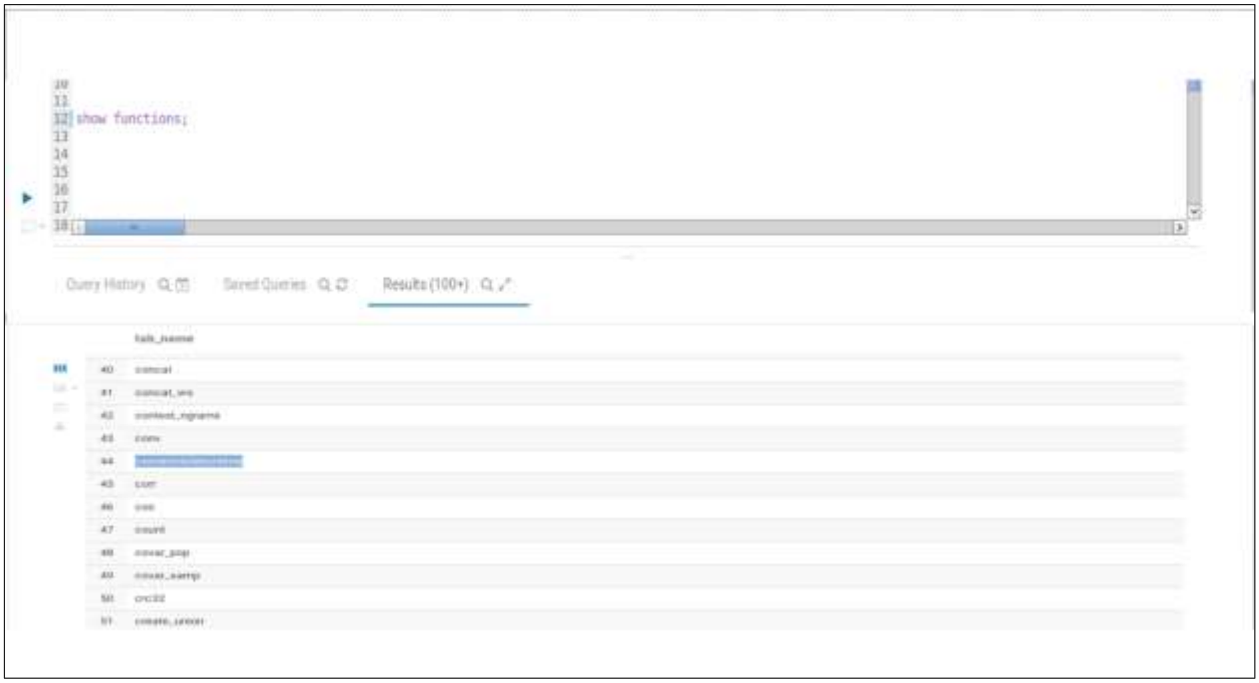
create temporary function ConvertViolationTime as 'com.upgrad.hive.udf.ViolationTime' ;



```
create temporary function ConvertViolationTime as
'com.upgrad.hive.udf.ViolationTime' ;
```

```
✓ Success
```

```
a few seconds ago   create temporary function ConvertViolationTime as 'com.upgrad.hive.udf.ViolationTime'
6 minutes ago     ✓ show functions
```



```
show functions;
```

	tab_name
40	concat
41	concat_ws
42	context_ngrams
43	conv
44	[illegible]
45	corr
46	cos
47	count
48	covar_pop
49	covar_samp
50	crc32
51	create_union

ASSUMPTIONS MADE IN THE UDF:

- Valid Format is considered as HHmmP / HHmmA
- If the violation time contains special characters such as ‘.+”’ or is the field is null then time is considered as HHmmP. i.e., violation time for that ticket is considered as 12:000 PM on that day.

  After conversion, it become as 12:00:00 (HH:mm:ss). Jar is attached.

GOOGLE DRIVE LINK FOR UDF:

https://drive.google.com/file/d/1Hv-Hi1qC7ZvdAekx32gzsvkAFTUkwY8Y/view?usp=sharing

CODE FOR UDF CONVERTVIOLATIONTIME:

```
package com.upgrad.hive.udf;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.hadoop.hive.ql.exec.UDF;
/**
 * This class converts the column violation time to hh:mm:ss 24 Hour format
 *
 * @param input of Type String
 * @return the time format in hh:mm:ss of 24 hr format
 *
 * Assumptions: If the violation time has any special characters, then
 * by default the input will be considered as 000P Which means the
 * violation time for that record is converted to 12:00:00 in hh:mm:ss
 */
public class ViolationTime extends UDF {

    public String evaluate(String input) {

          /* Checks for the special characters. If any exists, then by default the
time
           will be considered as 000P==>12:00:00 after conversion */
```

```
        if (input.contains(".") || input.contains("+") || input == null ||
input.trim().equals(""))
            input = "000P";
        String output = "00:00:00";
        DateFormat outputformat = new SimpleDateFormat("HH:mm:ss");

        /* If the input time contains "P" then it is considered as PM and 12 is
added to the first 2 digits of the input
          to convert in 24 hour format   */

        if (input.contains("P") || input.contains("p")) {
            String hour = input.substring(0, 2);
            int hour_format = Integer.parseInt(hour) + 12;
            String input1 = Integer.toString(hour_format) +
input.substring(2, 4);
            input = input1;
        }
        SimpleDateFormat df = new SimpleDateFormat("HHmm");
        Date date = null;
        try {
            //Parsing the date as per the outputformat HH:mm:ss
            date = df.parse(input);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        // Formatting of date and return
        output = outputformat.format(date);
        return output;
    }
    public static void main(String[] args) {
        ViolationTime obj = new ViolationTime();
        String test = "0143P";
        System.out.println(obj.evaluate(test));
        }
}
```

TABLES CREATED

STAGING TABLE:

This table is created to load the nyc parking violations data provided as is without using any UDF’s or in-built functions or filters.

CREATE STATEMENT FOR STAGING TABLE

```
create table NYC_VIOLATIONS_STAGING(`SummonsNumber` bigint,`PlateID`
string,`RegistrationState` string,`PlateType` string,`IssueDate` string,`ViolationCode`
int,`VehicleBodyType` string,`VehicleMake` string,`IssuingAgency` string,`StreetCode1`
string,`StreetCode2` string,`StreetCode3` string,`VehicleExpirationDate` bigint,`ViolationLocation`
string,`ViolationPrecinct` int,`IssuerPrecinct` int,`IssuerCode` int,`IssuerCommand`
string,`IssuerSquad` string,`ViolationTime` string,`TimeFirstObserved` string,`ViolationCounty`
string,`ViolationInFrontOfOrOpposite` string,`HouseNumber` string,`StreetName`
string,`IntersectingStreet` string,`DateFirstObserved` int,`LawSection` int,`SubDivision`
string,`ViolationLegalCode` string,`DaysParkingInEffect` string,`FromHoursInEffect`
string,`ToHoursInEffect` string,`VehicleColor` string,`UnregisteredVehicle` string,`VehicleYear`
int,`MeterNumber` string,`FeetFromCurb` int,`ViolationPostCode` string,`ViolationDescription`
string,`NoStandingorStoppingViolation` string,`HydrantViolation` string,`DoubleParkingViolation`
string) row format delimited fields terminated by ',' tblproperties ("skip.header.line.count"="1");
```

MASTER TABLE

- This table holds the raw data along with a few pseudo columns of 2017.
- Created managed table.
- **Reason:** Due to below reasons, I have used internal table for completing the assignment
  - External table is suggestible when the data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files. And when the data needs to remain in the underlying location even after a DROP TABLE.
  - Whereas internal /managed table is used when the data is temporary and when we want hive to completely manage the lifecycle of the table and data.
- All queries are performed on this table.
- Loading only 2017 data into this table.
- Additional columns added to analyze the data set are:

    - Issuedate is converted to to_date using unix_timestamp.
    - Violation Time which is HHmmP format is converted to 24 hour format of HH:mm:ss by using UDF mentioned above.
    - IssueMonth is added by using Month() function on Issuedate.
    - ViolationHour is added by using Hour() function on violation time.
- Partitioned by Month.
- Clustered by the violation hour and violation code.

  Reason: Since most of the queries are based on the violation code and the time, it is advisable to cluster by those columns.
- Also, it is clustered by 3 buckets .

  Reason: Because, More the number of buckets more the number of files and more the seek time while querying the data. Since the data set is of only 54903 records which is of few Mb's, It is not suggestible to increase the buckets numbers. Increasing the buckets lead to more number of small files.

CREATE STATEMENT FOR MASTER TABLE

```
CREATE TABLE  nyc_partitioned_buketed_orc (`SummonsNumber` bigint,`PlateID`
string,`RegistrationState` string,`PlateType` string,`IssueDate` string,`IssueDateCnv`
date,`ViolationCode` int,`VehicleBodyType` string,`VehicleMake` string,`IssuingAgency`
string,`StreetCode1` string,`StreetCode2` string,`StreetCode3` string,`VehicleExpirationDate`
bigint,`ViolationLocation`string,`ViolationPrecinct` int,`IssuerPrecinct` int,`IssuerCode`
int,`IssuerCommand` string,`IssuerSquad` string,`ViolationTime` string,`ViolationTImeCnv`
string,`TimeFirstObserved` string,`ViolationCounty` string,`ViolationInFrontOfOrOpposite`
string,`HouseNumber` string,`StreetName` string,`IntersectingStreet` string,`DateFirstObserved`
int,`LawSection` int,`SubDivision` string,`ViolationLegalCode` string,`DaysParkingInEffect `
string,`FromHoursInEffect` string,`ToHoursInEffect` string,`VehicleColor` string,`UnregisteredVehicle`
string,`VehicleYear` int,`MeterNumber` string,`FeetFromCurb` int,`ViolationPostCode`
string,`ViolationDescription` string,`NoStandingorStoppingViolation` string,`HydrantViolation`
string,`DoubleParkingViolation` string,`violationhour` int) partitioned by(issuemonth int) clustered by
(violationhour,ViolationCode) into 3 buckets ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS ORC tblproperties ("orc.compress"="ZLIB");
```

INSERT STATEMENT FOR MASTER TABLE

```
insert overwrite table nyc_partitioned_buketed_orc partition (issuemonth) select SummonsNumber,PlateID,RegistrationState,PlateType,issuedate,to_date(from_unixtime(UNIX_TIMESTAMP(issuedate,"MM/dd/yyyy"))),ViolationCode,VehicleBodyType,VehicleMake,IssuingAgency,StreetCode1,StreetCode2,StreetCode3,VehicleExpirationDate,ViolationLocation,ViolationPrecinct,IssuerPrecinct,IssuerCode,IssuerCommand,IssuerSquad,ViolationTime,ConvertViolationTime(ViolationTime),TimeFirstObserved,ViolationCounty,ViolationInFrontOfOrOpposite,HouseNumber,StreetName,IntersectingStreet,DateFirstObserved,LawSection,SubDivision,ViolationLegalCode,DaysParkingInEffect,FromHoursInEffect,ToHoursInEffect,VehicleColor,UnregisteredVehicle,VehicleYear,MeterNumber,FeetFromCurb,ViolationPostCode,ViolationDescription,NoStandingorStoppingViolation,HydrantViolation,DoubleParkingViolation,hour(ConvertViolationTime(ViolationTime)) as violationhour,month(to_date(from_unixtime(UNIX_TIMESTAMP(issuedate,"MM/dd/yyyy")))) as issuemonth from NYC_VIOLATIONS_STAGING  where year(to_date(from_unixtime(UNIX_TIMESTAMP(issuedate,"MM/dd/yyyy"))))=2017;
```

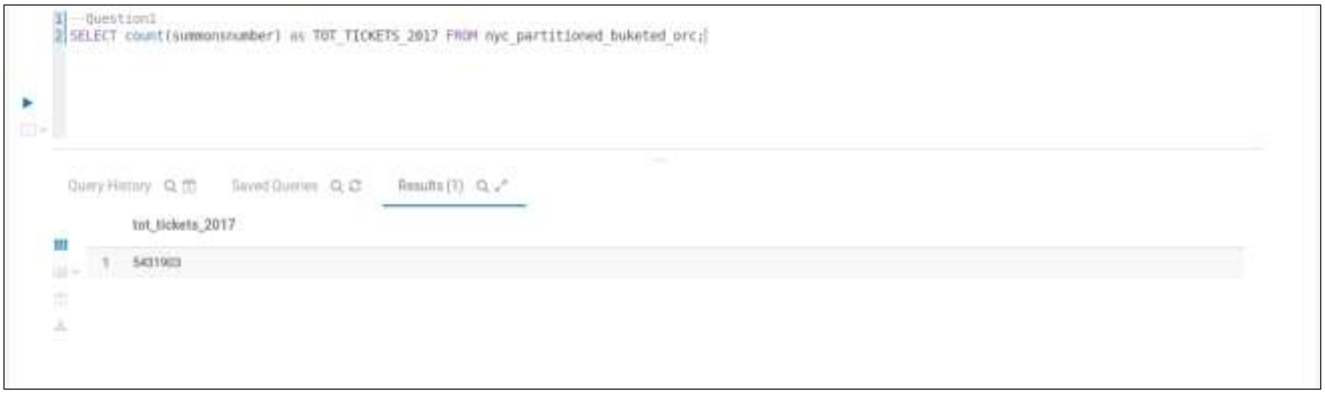
PART 1: EXAMINE THE DATA

1. FIND THE TOTAL NUMBER OF TICKETS FOR THE YEAR

QUERY:

```
SELECT count(summonsnumber) as TOT_TICKETS_2017 FROM nyc_partitioned_buketed_orc;
```

OUTPUT SCREEN SHOT:



```
--Question1
SELECT count(summonsnumber) as TOT_TICKETS_2017 FROM nyc_partitioned_buketed_orc;
```

	tot_tickets_2017
1	5431903

2. FIND OUT HOW MANY UNIQUE STATES THE CARS WHICH GOT PARKING TICKETS CAME FROM

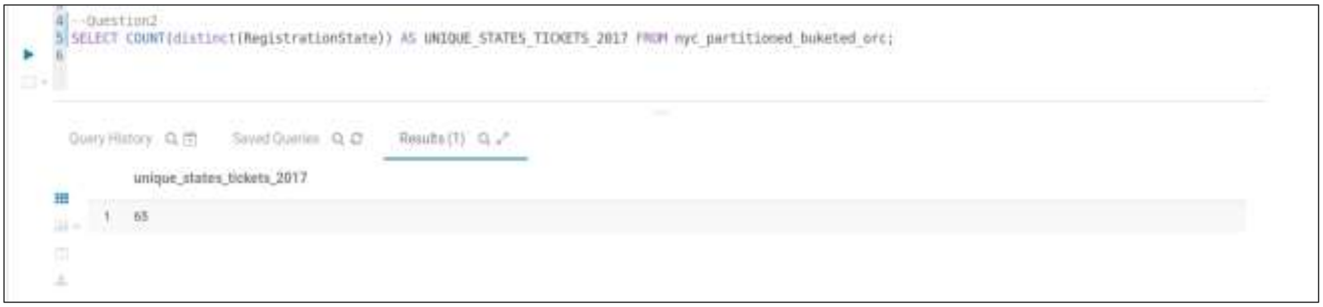
ASSUMPTION:

“99” is considered as a valid state since it has 16055 records with it.

QUERY:

SELECT COUNT(distinct(RegistrationState)) AS UNIQUE_STATES_TICKETS_2017 FROM nyc_partitioned_buketed_orc;

OUTPUT SCREESHOT:



```
--Question2
SELECT COUNT(distinct(RegistrationState)) AS UNIQUE_STATES_TICKETS_2017 FROM nyc_partitioned_buketed_orc;
```

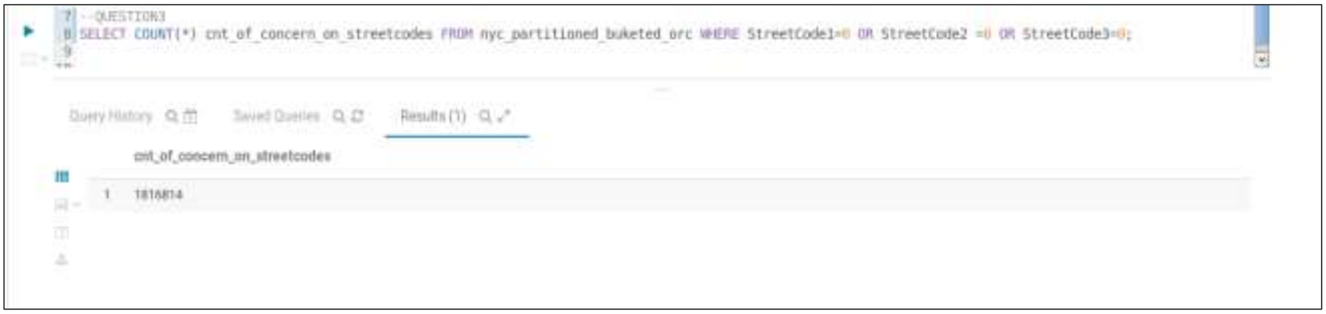
```
   unique_states_tickets_2017
1  65
```

3. SOME PARKING TICKETS DON’T HAVE ADDRESSES ON THEM, WHICH IS CAUSE FOR CONCERN. FIND OUT HOW MANY SUCH TICKETS THERE ARE

QUERY:

SELECT COUNT(*) cnt_of_concern_on_streetcodes FROM nyc_partitioned_buketed_orc WHERE StreetCode1=0 OR StreetCode2 =0 OR StreetCode3=0;

OUTPUT SCREEN SHOT:



```
--QUESTION3
SELECT COUNT(*) cnt_of_concern_on_streetcodes FROM nyc_partitioned_buketed_orc WHERE StreetCode1=0 OR StreetCode2 =0 OR StreetCode3=0;
```

```
   cnt_of_concern_on_streetcodes
1  1816814
```

PART2 : AGGREGATION TASKS

1. HOW OFTEN DOES EACH VIOLATION CODE OCCUR? TOP 5?

ASSUMPTION:

Violation code "0" considered to be invalid. Hence excluded from the query in where clause

STATISTICS:

Top 5 violation codes and their respective counts occurred are as shown in the table below.

top_5_violationcodes	count_of_violation_code_occured
21	768082
36	662765
38	542079
14	476660
20	319646

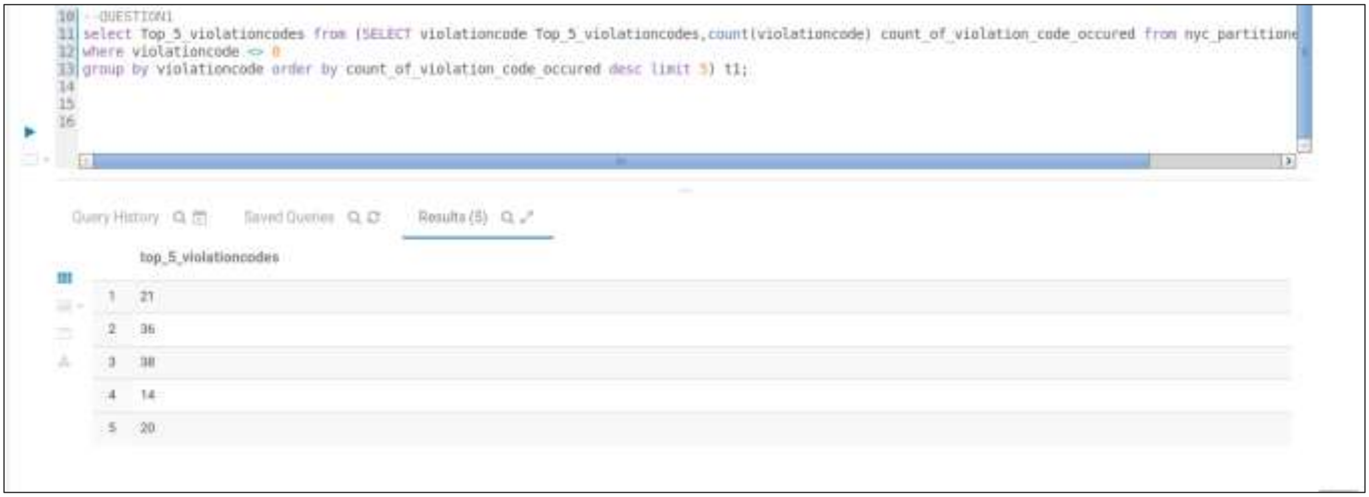
QUERY:

select Top_5_violationcodes from (SELECT violationcode Top_5_violationcodes,count(violationcode) count_of_violation_code_occured from nyc_partitioned_buketed_orc

where violationcode <> 0

group by violationcode order by count_of_violation_code_occured desc limit 5) t1;

OUTPUT SCRRENSHOT:



```
--QUESTION1
select Top_5_violationcodes from (SELECT violationcode Top_5_violationcodes,count(violationcode) count_of_violation_code_occured from nyc_partitione
where violationcode <> 0
group by violationcode order by count_of_violation_code_occured desc limit 5) t1;
```

```
   top_5_violationcodes
1  21
2  36
3  38
4  14
5  20
```

2.1. HOW OFTEN DOES EACH VEHICLE BODY TYPE GET A PARKING TICKET? TOP 5?

ASSUMPTION:

Vehicle body type as null is excluded from the query while fetching the top5 vehicle body types.

STATICTCS:

Top 5 vehicle body types and their respective counts occurred are as shown in the table below.

top_5_vehiclebodytypes	count_of_vehicle_body
SUBN	1883953
4DSD	1547307
VAN	724025
DELV	358982
SDN	194197

QUERY:

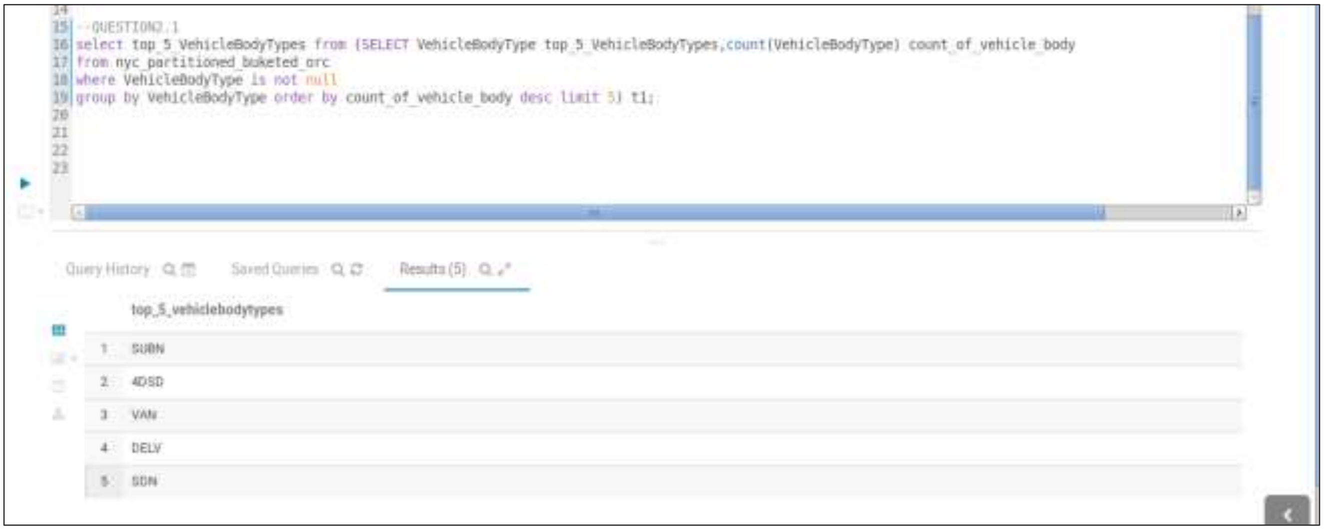
select top_5_VehicleBodyTypes from (SELECT VehicleBodyType top_5_VehicleBodyTypes,count(VehicleBodyType) count_of_vehicle_body

from nyc_partitioned_buketed_orc

where VehicleBodyType is not null

group by VehicleBodyType order by count_of_vehicle_body desc limit 5) t1;

OUTPUT:



```
--QUESTION2.1
select top_5_VehicleBodyTypes from (SELECT VehicleBodyType top_5_VehicleBodyTypes,count(VehicleBodyType) count_of_vehicle_body
from nyc_partitioned_buketed_orc
where VehicleBodyType is not null
group by VehicleBodyType order by count_of_vehicle_body desc limit 5) t1;
```

```
   top_5_vehiclebodytypes
1  SUBN
2  4DSD
3  VAN
4  DELV
5  SDN
```

2.2. HOW OFTEN DOES EACH VEHICLE MAKE TYPE GET A PARKING TICKET? TOP 5?

ASSUMPTION:

Vehicle Make as null is excluded from the query while fetching the top5 vehicle makes.

STATISCTICS:

Top 5 vehicle makes and their respective counts occurred are as shown in the table below.

top_5_vehiclemakes	count_of_vehicle_make
FORD	636842
TOYOT	605290
HONDA	538884
NISSA	462017
CHEVR	356032

QUERY:

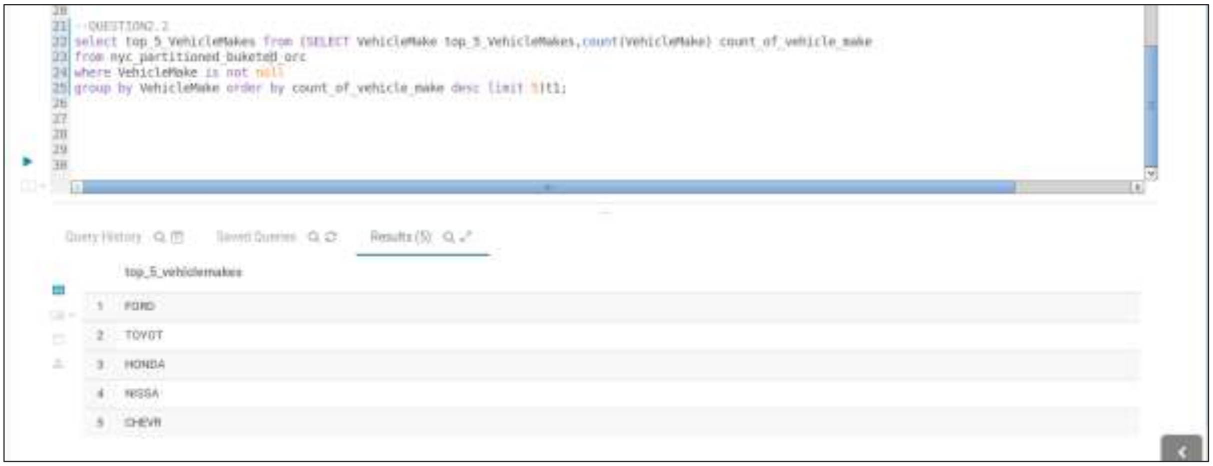
select top_5_VehicleMakes from (SELECT VehicleMake top_5_VehicleMakes,count(VehicleMake) count_of_vehicle_make

from nyc_partitioned_buketed_orc

where VehicleMake is not null

group by VehicleMake order by count_of_vehicle_make desc limit 5)t1;

OUTPUT SCREENSHOT:



```
--QUESTION2.2
select top_5_VehicleMakes from (SELECT VehicleMake top_5_VehicleMakes,count(VehicleMake) count_of_vehicle_make
from nyc_partitioned_buketed_orc
where VehicleMake is not null
group by VehicleMake order by count_of_vehicle_make desc limit 5)t1;
```

	top_5_vehiclemakes
1	FORD
2	TOYOT
3	HONDA
4	NISSA
5	CHEVR

3.1. FIND THE (5 HIGHEST) FREQUENCIES OF VIOLATING PRECINCTS

ASSUMPTION:

Violation Precinct as "0" is considered as invalid. Hence excluded from the query while fetching top 5 violation precincts.

STATISTICS:

Top 5 violation precincts and their respective counts occurred are as shown in the table below.

top_5_violationprecincts	count_of_violation_precinct
19	274443
14	203552
1	174702
18	169131
114	147444

QUERY

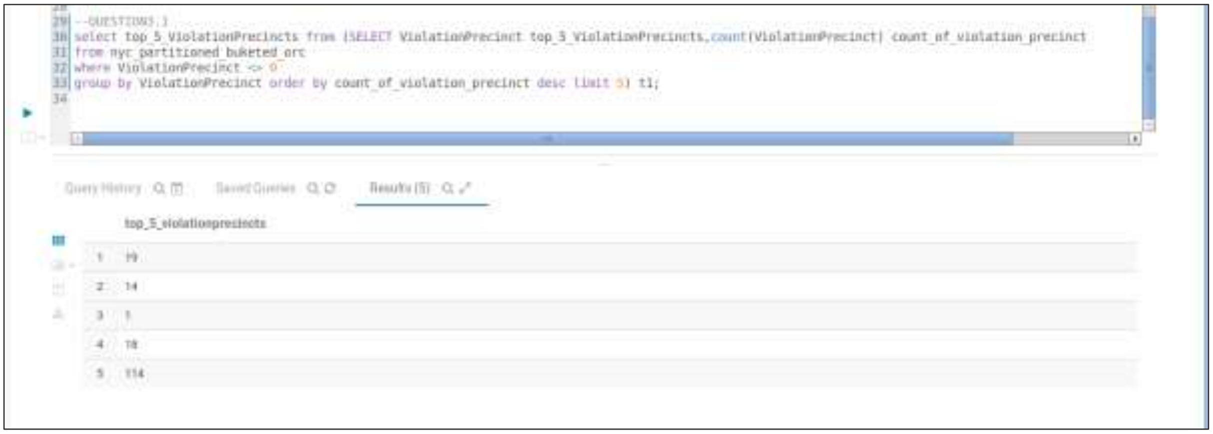
select top_5_ViolationPrecincts from (SELECT ViolationPrecinct top_5_ViolationPrecincts,count(ViolationPrecinct) count_of_violation_precinct

from nyc_partitioned_buketed_orc

where ViolationPrecinct <> 0

group by ViolationPrecinct order by count_of_violation_precinct desc limit 5) t1;

OUTPUT SCREENSHOT



```
--QUESTION3.1
select top_5_ViolationPrecincts from (SELECT ViolationPrecinct top_5_ViolationPrecincts,count(ViolationPrecinct) count_of_violation_precinct
from nyc_partitioned_buketed_orc
where ViolationPrecinct <> 0
group by ViolationPrecinct order by count_of_violation_precinct desc limit 5) t1;
```

	top_5_violationprecincts
1	19
2	14
3	1
4	18
5	114

3.2. FIND THE (5 HIGHEST) FREQUENCIES OF ISSUER PRECINCTS

ASSUMPTION:

Issuer Precinct as “0” is considered as invalid. Hence excluded from the query while fetching top 5 Issuer precincts.

STATISTICS:

Top 5 issuer precincts and their respective counts occurred are as shown in the table below.

top_5_issuerprecincts	count_of_issuer_precinct
19	266959
14	200494
1	168740
18	162994
114	144054

QUERY

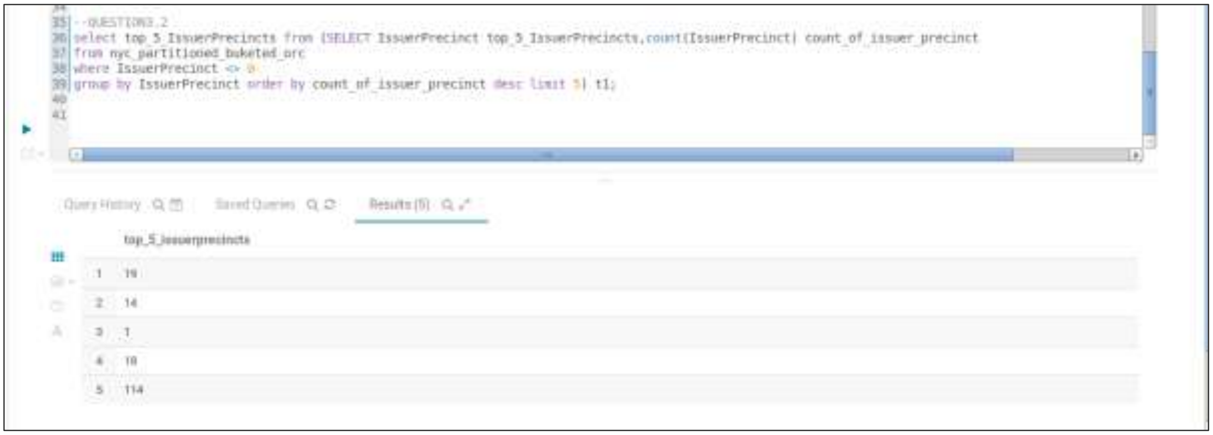
select top_5_IssuerPrecincts from (SELECT IssuerPrecinct top_5_IssuerPrecincts,count(IssuerPrecinct) count_of_issuer_precinct

from nyc_partitioned_buketed_orc

where IssuerPrecinct <> 0

group by IssuerPrecinct order by count_of_issuer_precinct desc limit 5) t1;

OUTPUT SCREENSHOT:



```
--QUESTION3.2
select top_5_IssuerPrecincts from (SELECT IssuerPrecinct top_5_IssuerPrecincts,count(IssuerPrecinct) count_of_issuer_precinct
from nyc_partitioned_buketed_orc
where IssuerPrecinct <> 0
group by IssuerPrecinct order by count_of_issuer_precinct desc limit 5) t1;
```

	top_5_issuerprecincts
1	19
2	14
3	1
4	18
5	114

4. FIND THE VIOLATION CODE FREQUENCY ACROSS 3 PRECINCTS WHICH HAVE ISSUED THE MOST NUMBER OF TICKETS

STATISTICS

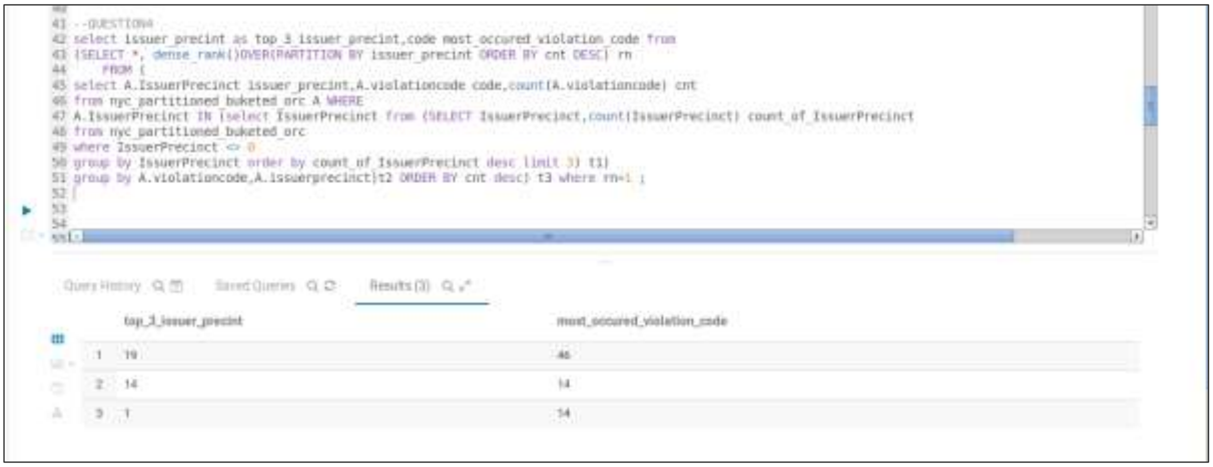
Top 3 issuer precincts, its most occurred violation and corresponding count of occurrence are as shown in the table below.

top_3_issuer_precint	most_occured_violation_code	no_of_times_occured
19	46	48444
14	14	45036
1	14	38354

QUERY

```
select issuer_precint,code most_occured_violation_code,cnt no_of_times_occured from

(SELECT *, dense_rank()OVER(PARTITION BY issuer_precint ORDER BY cnt DESC) rn

  FROM (

select A.IssuerPrecinct issuer_precint,A.violationcode code,count(A.violationcode) cnt

from nyc_partitioned_buketed_orc A WHERE

A.IssuerPrecinct IN (select IssuerPrecinct from (SELECT IssuerPrecinct,count(IssuerPrecinct) count_of_IssuerPrecinct

from nyc_partitioned_buketed_orc

where IssuerPrecinct <> 0

group by IssuerPrecinct order by count_of_IssuerPrecinct desc limit 3) t1)

group by A.violationcode,A.issuerprecinct)t2) t3 where rn=1 ;
```

OUTPUT SCREENSHOT:



```
--QUESTION4
select issuer_precint as top_3_issuer_precint,code most_occured_violation_code from
(SELECT *, dense_rank()OVER(PARTITION BY issuer_precint ORDER BY cnt DESC) rn
    FROM (
select A.IssuerPrecinct issuer_precint,A.violationcode code,count(A.violationcode) cnt
from nyc_partitioned_buketed_orc A WHERE
A.IssuerPrecinct IN (select IssuerPrecinct from (SELECT IssuerPrecinct,count(IssuerPrecinct) count_of_IssuerPrecinct
from nyc_partitioned_buketed_orc
where IssuerPrecinct <> 0
group by IssuerPrecinct order by count_of_IssuerPrecinct desc limit 3) t1)
group by A.violationcode,A.issuerprecinct)t2 ORDER BY cnt desc) t3 where rn=1 ;
```

	top_3_issuer_precint	most_occured_violation_code
1	19	46
2	14	14
3	1	14

5. FIND OUT THE PROPERTIES OF PARKING VIOLATIONS ACROSS DIFFERENT TIMES OF THE DAY

DESCRIPTION:

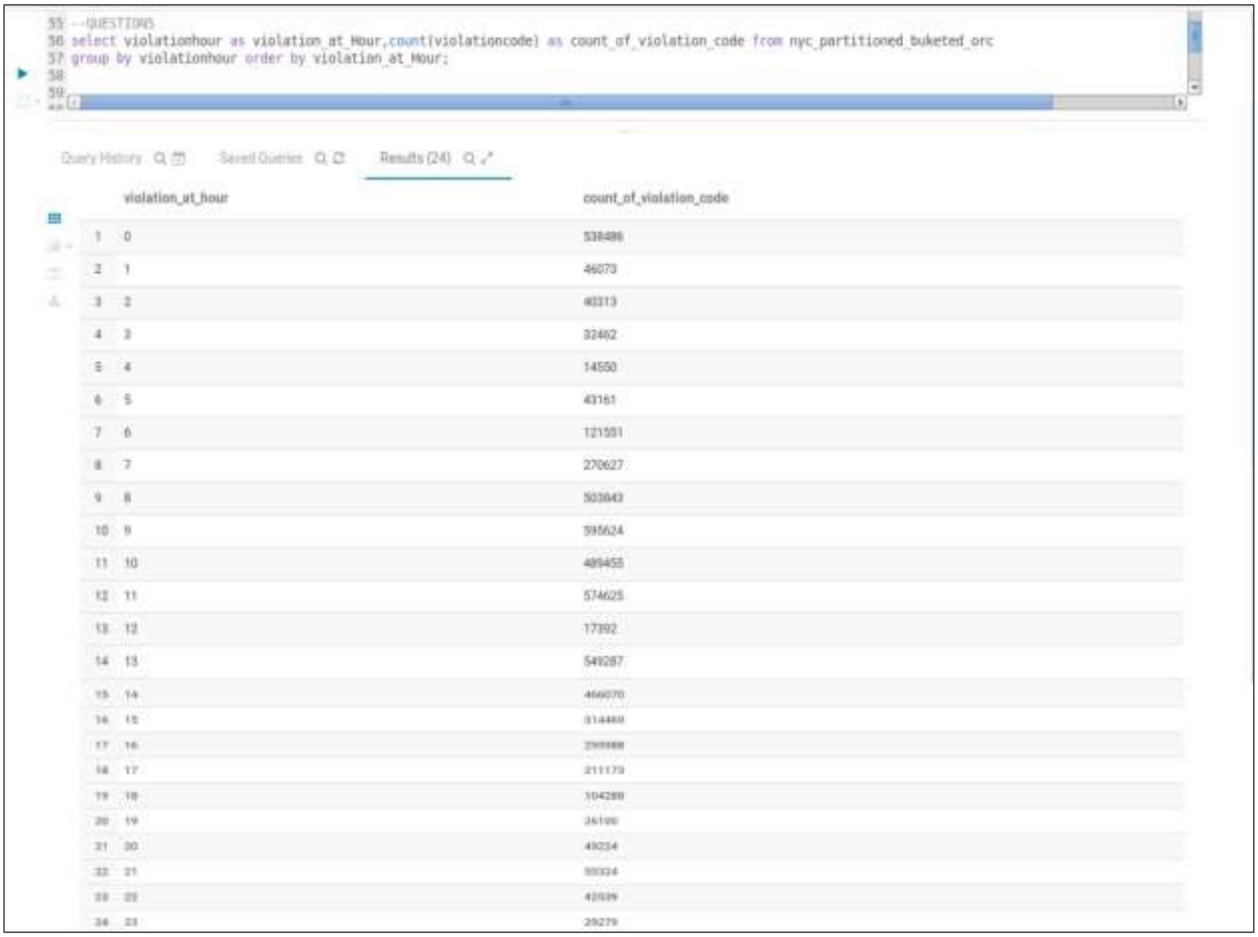
ViolationHour column used in below query is derived by applying UDF on violation time column.

QUERY:

select violationhour as violation_at_Hour,count(violationcode) as count_of_violation_code from nyc_partitioned_buketed_orc

group by violationhour order by violation_at_Hour;

OUTPUT SCREENSHOT:



```
--QUESTIONS
select violationhour as violation_at_Hour,count(violationcode) as count_of_violation_code from nyc_partitioned_buketed_orc
group by violationhour order by violation_at_Hour;
```

	violation_at_hour	count_of_violation_code
1	0	538486
2	1	46073
3	2	40313
4	3	32462
5	4	14550
6	5	43161
7	6	121551
8	7	270627
9	8	503843
10	9	595624
11	10	489455
12	11	574625
13	12	17392
14	13	549287
15	14	466070
16	15	314469
17	16	295988
18	17	211173
19	18	104289
20	19	26100
21	20	49234
22	21	55324
23	22	42539
24	23	29279

6. DIVIDE 24 HOURS INTO 6 EQUAL DISCRETE BINS OF TIME. FOR EACH OF THESE GROUPS, FIND THE 3 MOST COMMONLY OCCURRING VIOLATIONS

DESCRIPTION:

ViolationHour column used in below query for forming the bins is derived by applying UDF on violation time column.

STATISTICS:

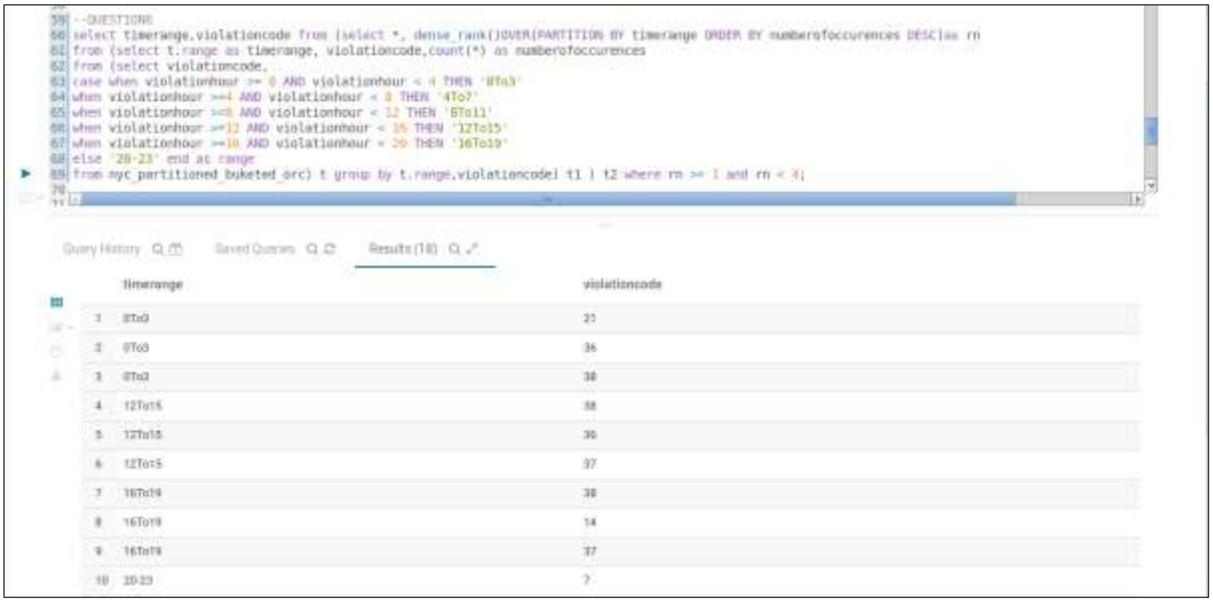
Top 3 violation codes for each of the bins and their respective count of occurrences are shown in the below table.

timerange	violationcode	numberofoccurences
0To3	21	107077
0To3	36	101991
0To3	38	56204
12To15	38	184829
12To15	36	184293
12To15	37	130692
16To19	38	102856
16To19	14	75902
16To19	37	70345
20-23	7	26293
20-23	40	22338
20-23	14	21045
4To7	14	74113
4To7	40	60652
4To7	21	57898
8To11	21	598055
8To11	36	348165
8To11	38	176570

QUERY:

```
select timerange,violationcode from (select *, dense_rank()OVER(PARTITION BY timerange ORDER BY numberofoccurences DESC)as rn

from (select t.range as timerange, violationcode,count(*) as numberofoccurences

from (select violationcode,

case when violationhour >= 0 AND violationhour < 4 THEN '0To3'

when violationhour >=4 AND violationhour < 8 THEN '4To7'

when violationhour >=8 AND violationhour < 12 THEN '8To11'

when violationhour >=12 AND violationhour < 16 THEN '12To15'

when violationhour >=16 AND violationhour < 20 THEN '16To19'  else '20-23' end as range

from nyc_partitioned_buketed_orc) t group by t.range,violationcode) t1 ) t2 where rn >= 1 and rn < 4;
```

OUTPUT SCREENSHOT:



```
--QUESTION6
select timerange,violationcode from (select *, dense_rank()OVER(PARTITION BY timerange ORDER BY numberofoccurences DESC)as rn
from (select t.range as timerange, violationcode,count(*) as numberofoccurences
from (select violationcode,
case when violationhour >= 0 AND violationhour < 4 THEN '0To3'
when violationhour >=4 AND violationhour < 8 THEN '4To7'
when violationhour >=8 AND violationhour < 12 THEN '8To11'
when violationhour >=12 AND violationhour < 16 THEN '12To15'
when violationhour >=16 AND violationhour < 20 THEN '16To19'
else '20-23' end as range
from nyc_partitioned_buketed_orc) t group by t.range,violationcode) t1 ) t2 where rn >= 1 and rn < 4;
```

```
    timerange    violationcode
1   0To3         21
2   0To3         36
3   0To3         38
4   12To15       38
5   12To15       36
6   12To15       37
7   16To19       38
8   16To19       14
9   16To19       37
10  20-23        7
```

7. FOR THE 3 MOST COMMONLY OCCURRING VIOLATION CODES, FIND THE MOST COMMON TIMES OF DAY

STATISTICS:

Top3 violation codes, it corresponding count and the most common time it has occurred is shown in the below table.

most_common_time_of_Day	violationcode	cnt_of_occurence
8To11	21	598055
8To11	36	348165
12To15	38	184829

QUERY:

select violationcode,most_common_time_of_Day from (select timerange as most_common_time_of_Day,violationcode,numberofoccurences from

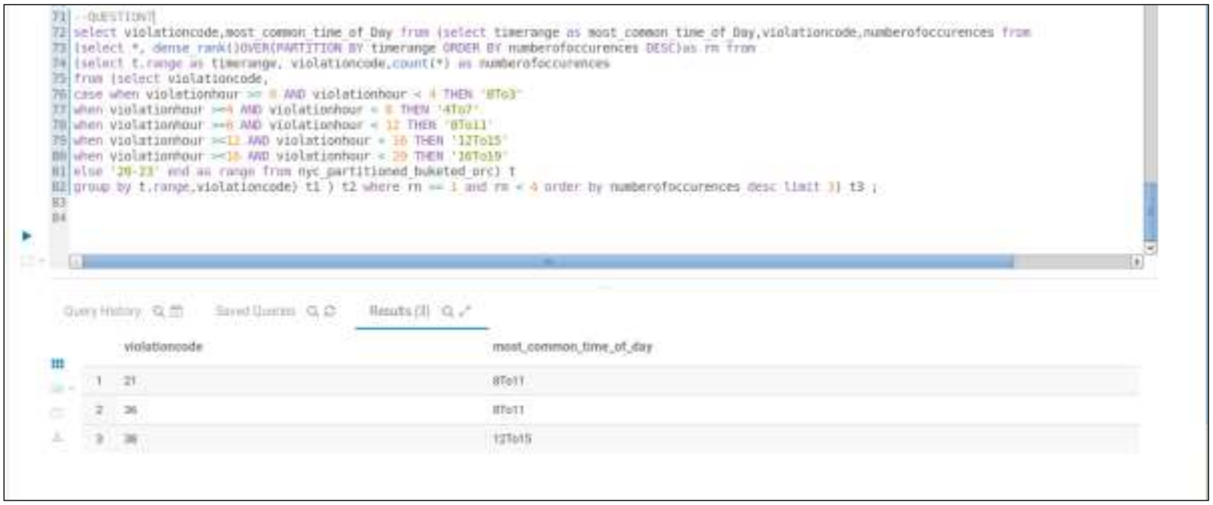
(select *, dense_rank()OVER(PARTITION BY timerange ORDER BY numberofoccurences DESC)as rn from

(select t.range as timerange, violationcode,count(*) as numberofoccurences

from (select violationcode,

```
case when violationhour >= 0 AND violationhour < 4 THEN '0To3'

when violationhour >=4 AND violationhour < 8 THEN '4To7'

when violationhour >=8 AND violationhour < 12 THEN '8To11'

when violationhour >=12 AND violationhour < 16 THEN '12To15'

when violationhour >=16 AND violationhour < 20 THEN '16To19'

else '20-23' end as range from nyc_partitioned_buketed_orc) t

group by t.range,violationcode) t1 ) t2 where rn >= 1 and rn < 4 order by numberofoccurences desc limit 3) t3 ;
```

OUTPUT SCREENSHOT:



```
--QUESTION7
select violationcode,most_common_time_of_Day from (select timerange as most_common_time_of_Day,violationcode,numberofoccurences from
(select *, dense_rank()OVER(PARTITION BY timerange ORDER BY numberofoccurences DESC)as rn from
(select t.range as timerange, violationcode,count(*) as numberofoccurences
from (select violationcode,
case when violationhour >= 0 AND violationhour < 4 THEN '0To3'
when violationhour >=4 AND violationhour < 8 THEN '4To7'
when violationhour >=8 AND violationhour < 12 THEN '8To11'
when violationhour >=12 AND violationhour < 16 THEN '12To15'
when violationhour >=16 AND violationhour < 20 THEN '16To19'
else '20-23' end as range from nyc_partitioned_buketed_orc) t
group by t.range,violationcode) t1 ) t2 where rn >= 1 and rn < 4 order by numberofoccurences desc limit 3) t3 ;
```

	violationcode	most_common_time_of_day
1	21	8To11
2	36	8To11
3	38	12To15

8.1. FIND FREQUENCIES OF TICKETS FOR EACH SEASON

QUERY:

select season, count(*) as nooftickets from (select summonsnumber,

case when issuemonth >= 3 AND issuemonth <= 5 THEN 'Spring'

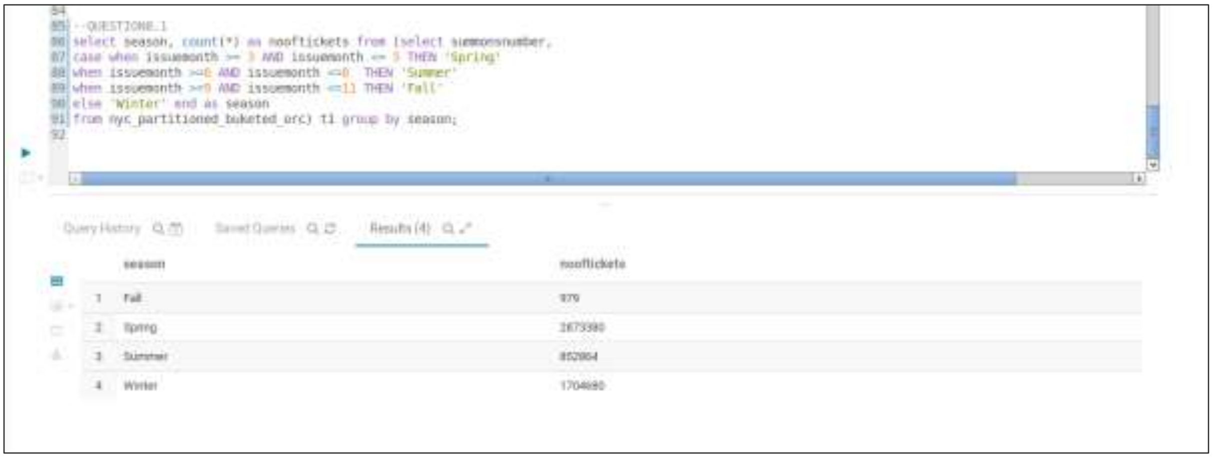
when issuemonth >=6 AND issuemonth <=8 THEN 'Summer'

when issuemonth >=9 AND issuemonth <=11 THEN 'Fall'

else 'Winter' end as season

from nyc_partitioned_buketed_orc) t1 group by season;

OUTPUT SCREENSHOT:



```
--QUESTIONE.1
select season, count(*) as nooftickets from (select summonsnumber,
case when issuemonth >= 3 AND issuemonth <= 5 THEN 'Spring'
when issuemonth >=6 AND issuemonth <=8  THEN 'Summer'
when issuemonth >=9 AND issuemonth <=11 THEN 'Fall'
else 'Winter' end as season
from nyc_partitioned_buketed_orc) t1 group by season;
```

	season	nooftickets
1	Fall	979
2	Spring	2673380
3	Summer	852864
4	Winter	1704690

8.2. FIND THE 3 MOST COMMON VIOLATIONS FOR EACH OF THESE SEASONS

STATISTICS:

season	violationcode	numberofoccurences	rn
Fall	46	231	1
Fall	21	128	2
Fall	40	116	3
Spring	21	402424	1
Spring	36	344834	2

Spring	38	271167	3
Summer	21	127350	1
Summer	36	96663	2
Summer	38	83518	3
Winter	21	238180	1
Winter	36	221268	2
Winter	38	187386	3

QUERY:

select season,violationcode from (select *, rank()OVER(PARTITION BY season ORDER BY numberofoccurences DESC)as rn from

(select t.season as season, violationcode,count(*) as numberofoccurences from (select violationcode, case when issuemonth >= 3 AND issuemonth <= 5 THEN 'Spring'

when issuemonth >=6 AND issuemonth <=8 THEN 'Summer'

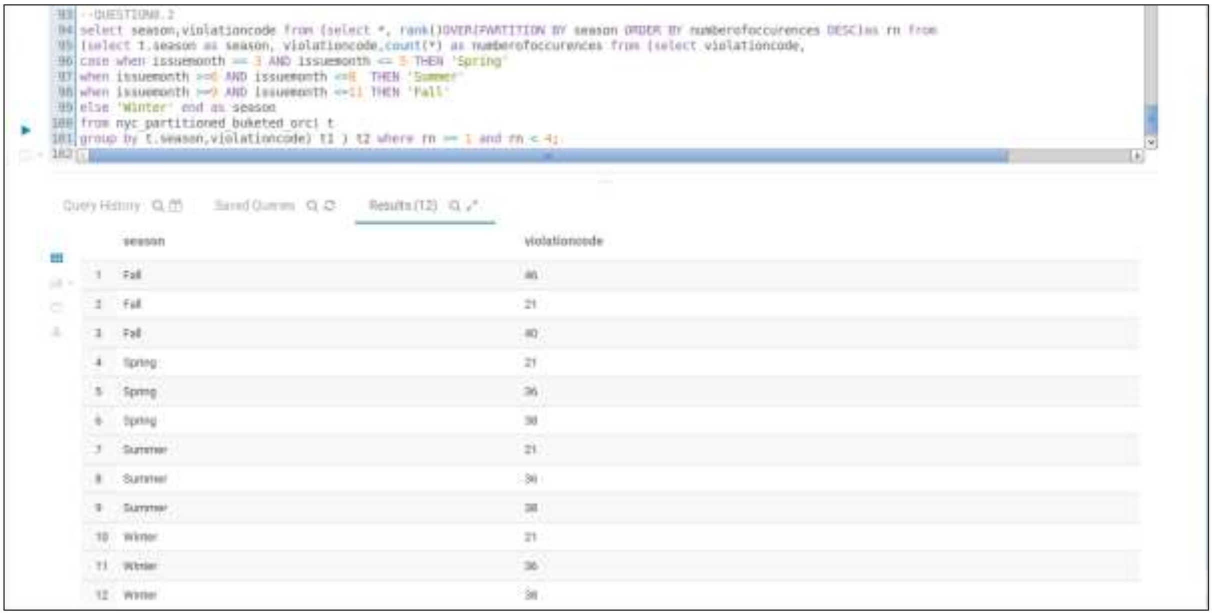
when issuemonth >=9 AND issuemonth <=11 THEN 'Fall'

else 'Winter' end as season

from nyc_partitioned_buketed_orc) t

group by t.season,violationcode) t1) t2 where rn >= 1 and rn < 4;

OUTPUT SCREENSHOT:



```
--QUESTION#.2
select season,violationcode from (select *, rank()OVER(PARTITION BY season ORDER BY numberofoccurences DESC)as rn from
(select t.season as season, violationcode,count(*) as numberofoccurences from (select violationcode,
case when issuemonth >= 3 AND issuemonth <= 5 THEN 'Spring'
when issuemonth >=6 AND issuemonth <=8  THEN 'Summer'
when issuemonth >=9 AND issuemonth <=11 THEN 'Fall'
else 'Winter' end as season
from nyc_partitioned_buketed_orc1 t
group by t.season,violationcode) t1 ) t2 where rn >= 1 and rn < 4;
```

	season	violationcode
1	Fall	46
2	Fall	21
3	Fall	40
4	Spring	21
5	Spring	36
6	Spring	38
7	Summer	21
8	Summer	36
9	Summer	38
10	Winter	21
11	Winter	36
12	Winter	38