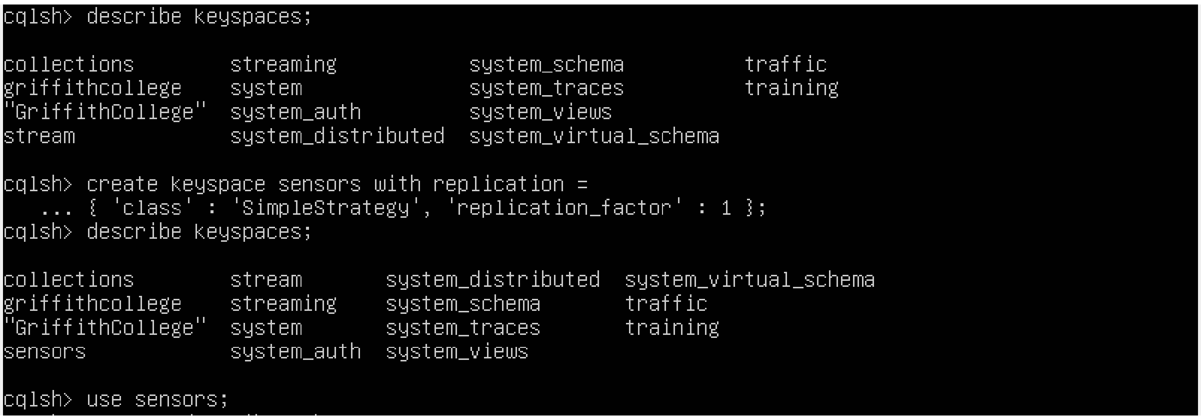
Big Data Management | Irish M50 Traffic Sensors Data

Extract Transform Load | Apache Spark | Cassandra

This document explores ETL process using Apache Spark Data Frame API to perform data transformation queries and store the results in Cassandra Data Structures. Irish Road Network has installed sensors on specific location that collect vehicle data including location, type, speed, weight and more. This data is stored in atomic form.

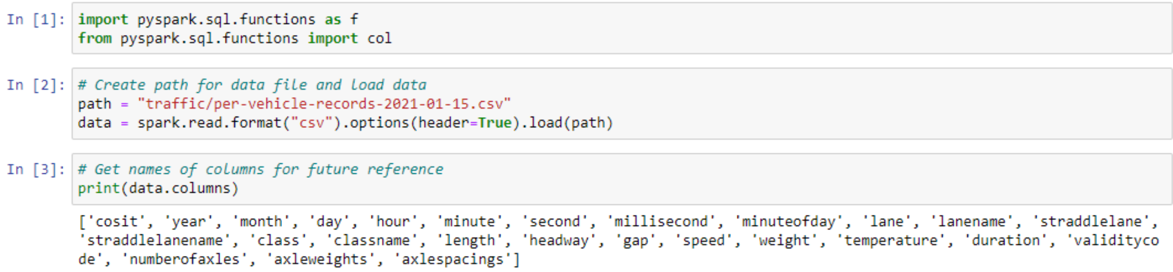
Dataset: Irish M50 Traffic Sensors Data

Create a new keyspace named sensor.



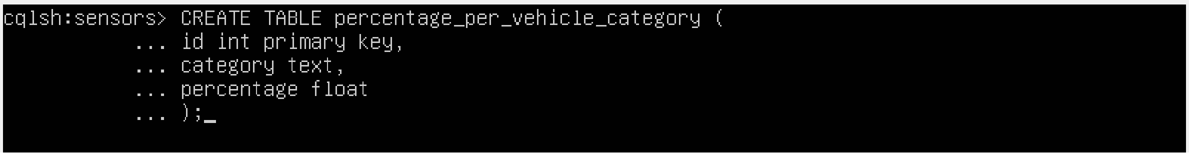
```
cqlsh> describe keyspaces;

collections        streaming          system_schema          traffic
griffithcollege    system             system_traces          training
"GriffithCollege"  system_auth        system_views
stream             system_distributed system_virtual_schema

cqlsh> create keyspace sensors with replication =
   ... { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
cqlsh> describe keyspaces;

collections        stream       system_distributed  system_virtual_schema
griffithcollege    streaming    system_schema       traffic
"GriffithCollege"  system       system_traces       training
sensors            system_auth  system_views

cqlsh> use sensors;
```

Load dataset.



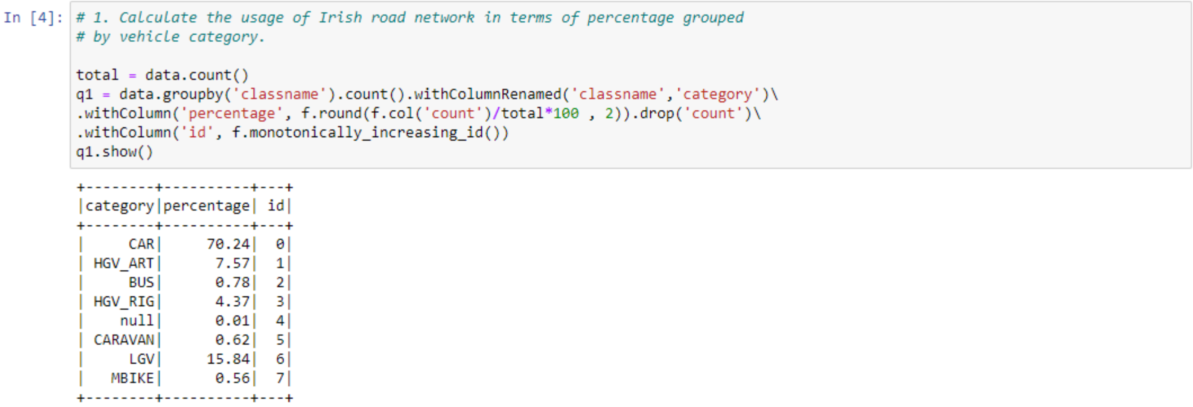
```
In [1]: import pyspark.sql.functions as f
        from pyspark.sql.functions import col

In [2]: # Create path for data file and load data
        path = "traffic/per-vehicle-records-2021-01-15.csv"
        data = spark.read.format("csv").options(header=True).load(path)

In [3]: # Get names of columns for future reference
        print(data.columns)

        ['cosit', 'year', 'month', 'day', 'hour', 'minute', 'second', 'millisecond', 'minuteofday', 'lane', 'lanename', 'straddlelane',
        'straddlelanename', 'class', 'classname', 'length', 'headway', 'gap', 'speed', 'weight', 'temperature', 'duration', 'validityco
        de', 'numberofaxles', 'axleweights', 'axlespacings']
```

Create data structure in Cassandra.



```
cqlsh:sensors> CREATE TABLE percentage_per_vehicle_category (
           ... id int primary key,
           ... category text,
           ... percentage float
           ... );_
```

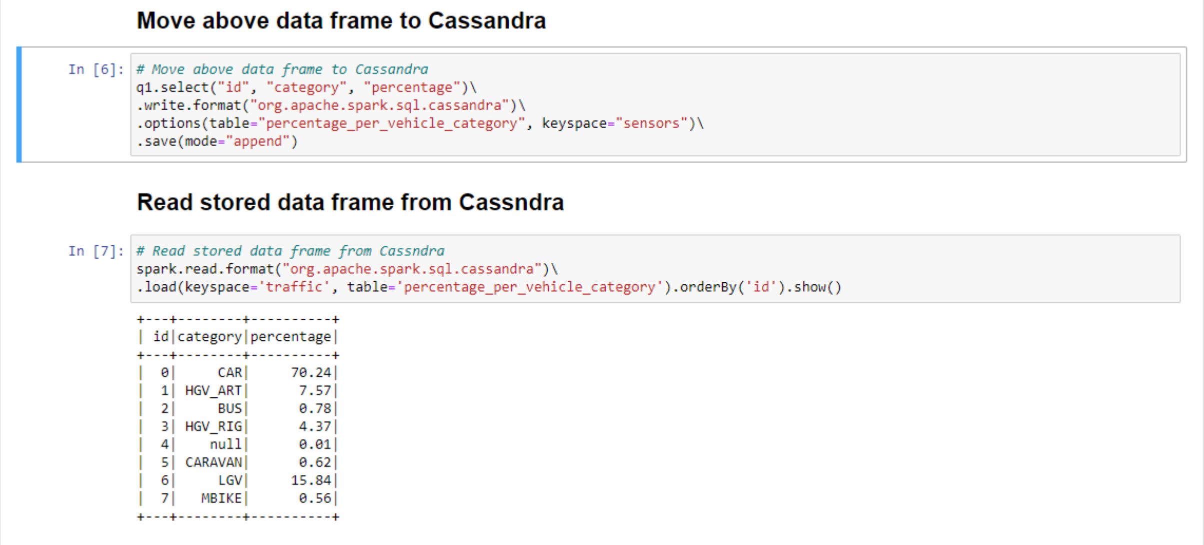
Perform Computation in Pyspark.



```
In [4]: # 1. Calculate the usage of Irish road network in terms of percentage grouped
        # by vehicle category.

        total = data.count()
        q1 = data.groupby('classname').count().withColumnRenamed('classname','category')\
        .withColumn('percentage', f.round(f.col('count')/total*100 , 2)).drop('count')\
        .withColumn('id', f.monotonically_increasing_id())
        q1.show()
```

```
+--------+----------+---+
|category|percentage| id|
+--------+----------+---+
|     CAR|     70.24|  0|
| HGV_ART|      7.57|  1|
|     BUS|      0.78|  2|
| HGV_RIG|      4.37|  3|
|    null|      0.01|  4|
| CARAVAN|      0.62|  5|
|     LGV|     15.84|  6|
|   MBIKE|      0.56|  7|
+--------+----------+---+
```

Store dataframe to Cassandra and Check.



Move above data frame to Cassandra

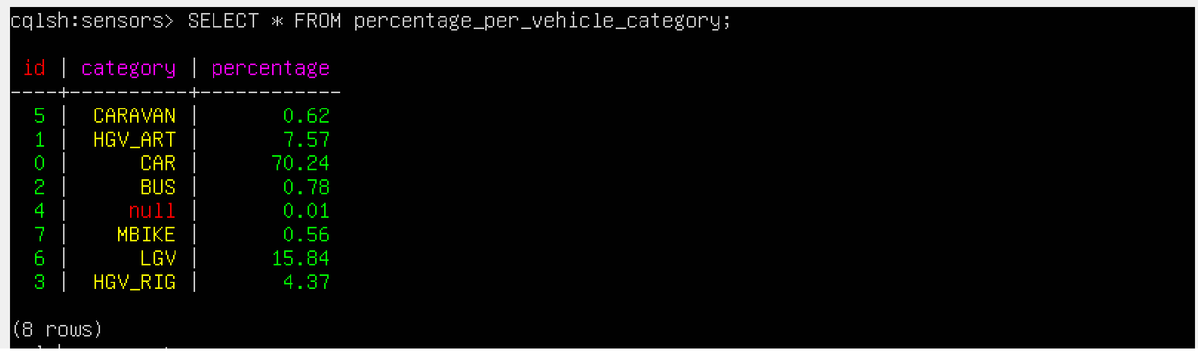
```
In [6]: # Move above data frame to Cassandra
        q1.select("id", "category", "percentage")\
        .write.format("org.apache.spark.sql.cassandra")\
        .options(table="percentage_per_vehicle_category", keyspace="sensors")\
        .save(mode="append")
```

Read stored data frame from Cassndra

```
In [7]: # Read stored data frame from Cassndra
        spark.read.format("org.apache.spark.sql.cassandra")\
        .load(keyspace='traffic', table='percentage_per_vehicle_category').orderBy('id').show()
```

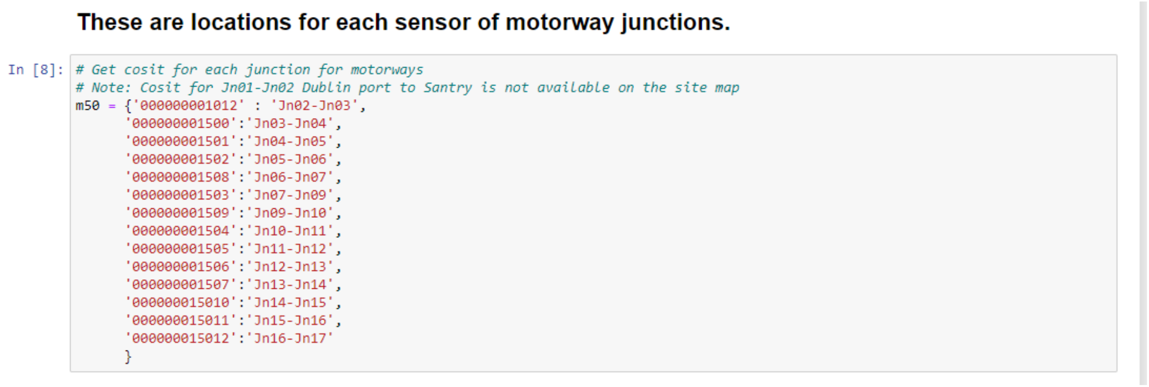
```
+---+--------+----------+
| id|category|percentage|
+---+--------+----------+
|  0|     CAR|     70.24|
|  1| HGV_ART|      7.57|
|  2|     BUS|      0.78|
|  3| HGV_RIG|      4.37|
|  4|    null|      0.01|
|  5| CARAVAN|      0.62|
|  6|     LGV|     15.84|
|  7|   MBIKE|      0.56|
+---+--------+----------+
```

Checking in Cassandra.



```
cqlsh:sensors> SELECT * FROM percentage_per_vehicle_category;

 id | category | percentage
----+----------+------------
  5 |  CARAVAN |       0.62
  1 |  HGV_ART |       7.57
  0 |      CAR |      70.24
  2 |      BUS |       0.78
  4 |     null |       0.01
  7 |    MBIKE |       0.56
  6 |      LGV |      15.84
  3 |  HGV_RIG |       4.37

(8 rows)
```

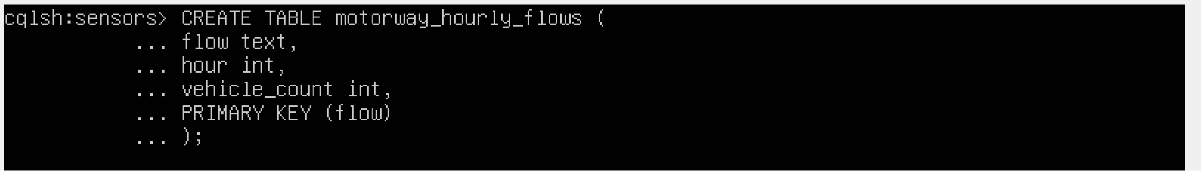
Create location dictionaries with sensor codes.



These are locations for each sensor of motorway junctions.

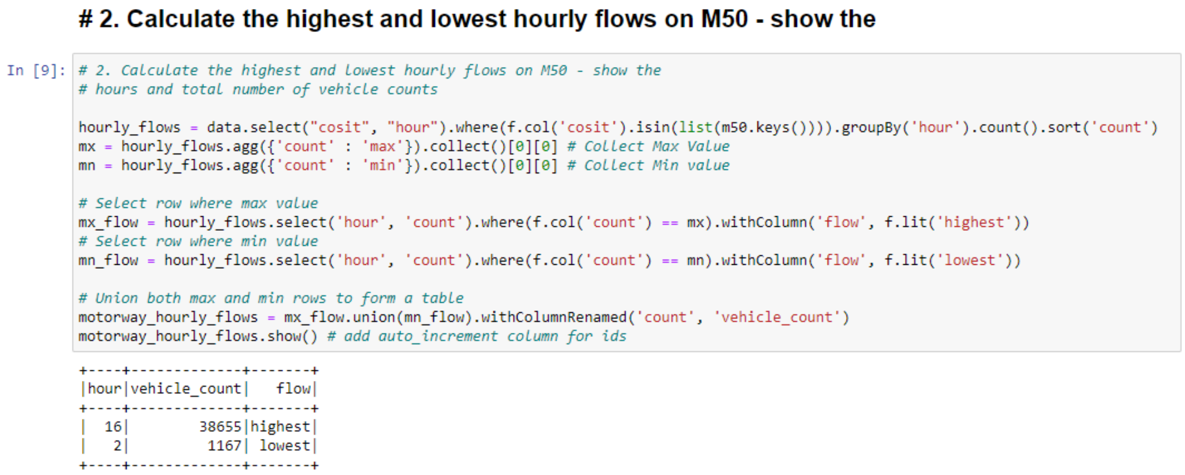
```
In [8]: # Get cosit for each junction for motorways
        # Note: Cosit for Jn01-Jn02 Dublin port to Santry is not available on the site map
        m50 = {'000000001012' : 'Jn02-Jn03',
               '000000001500':'Jn03-Jn04',
               '000000001501':'Jn04-Jn05',
               '000000001502':'Jn05-Jn06',
               '000000001508':'Jn06-Jn07',
               '000000001503':'Jn07-Jn09',
               '000000001509':'Jn09-Jn10',
               '000000001504':'Jn10-Jn11',
               '000000001505':'Jn11-Jn12',
               '000000001506':'Jn12-Jn13',
               '000000001507':'Jn13-Jn14',
               '000000015010':'Jn14-Jn15',
               '000000015011':'Jn15-Jn16',
               '000000015012':'Jn16-Jn17'
              }
```

Create cassndra structure for motorway hourly flows.



```
cqlsh:sensors> CREATE TABLE motorway_hourly_flows (
           ... flow text,
           ... hour int,
           ... vehicle_count int,
           ... PRIMARY KEY (flow)
           ... );
```

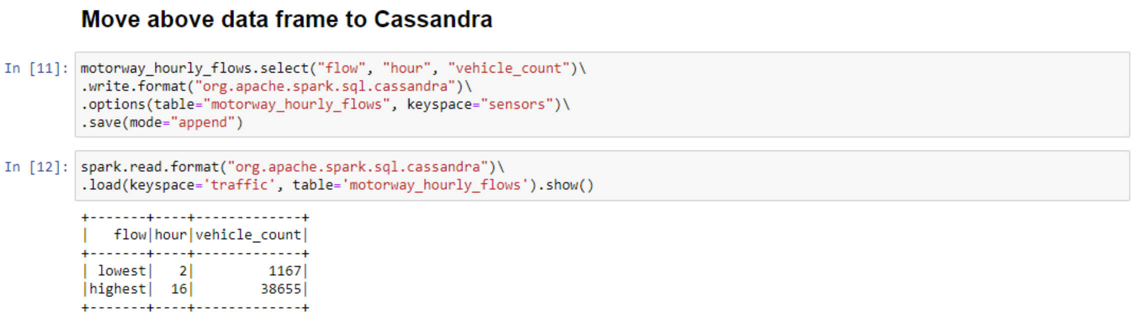
Perform computation.



2. Calculate the highest and lowest hourly flows on M50 - show the

```
In [9]: # 2. Calculate the highest and lowest hourly flows on M50 - show the
        # hours and total number of vehicle counts

        hourly_flows = data.select("cosit", "hour").where(f.col('cosit').isin(list(m50.keys()))).groupBy('hour').count().sort('count')
        mx = hourly_flows.agg({'count' : 'max'}).collect()[0][0] # Collect Max Value
        mn = hourly_flows.agg({'count' : 'min'}).collect()[0][0] # Collect Min value

        # Select row where max value
        mx_flow = hourly_flows.select('hour', 'count').where(f.col('count') == mx).withColumn('flow', f.lit('highest'))
        # Select row where min value
        mn_flow = hourly_flows.select('hour', 'count').where(f.col('count') == mn).withColumn('flow', f.lit('lowest'))

        # Union both max and min rows to form a table
        motorway_hourly_flows = mx_flow.union(mn_flow).withColumnRenamed('count', 'vehicle_count')
        motorway_hourly_flows.show() # add auto_increment column for ids

        +----+-------------+-------+
        |hour|vehicle_count|   flow|
        +----+-------------+-------+
        |  16|        38655|highest|
        |   2|         1167| lowest|
        +----+-------------+-------+
```

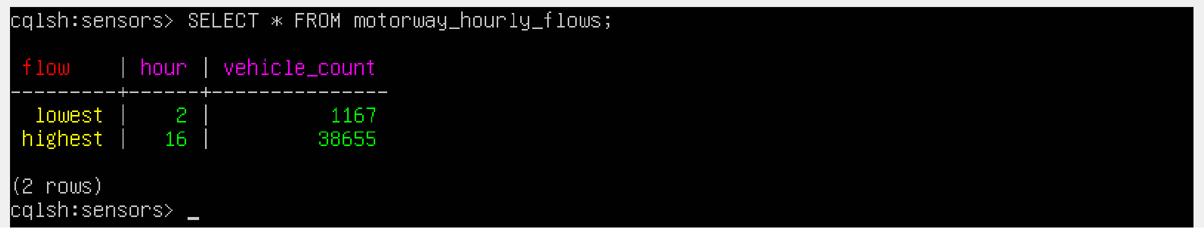
Move dataframe to Cassandra and check.



Move above data frame to Cassandra

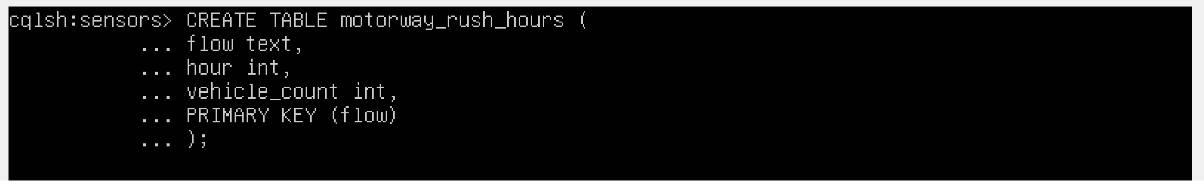
```
In [11]: motorway_hourly_flows.select("flow", "hour", "vehicle_count")\
         .write.format("org.apache.spark.sql.cassandra")\
         .options(table="motorway_hourly_flows", keyspace="sensors")\
         .save(mode="append")

In [12]: spark.read.format("org.apache.spark.sql.cassandra")\
         .load(keyspace='traffic', table='motorway_hourly_flows').show()

         +-------+----+-------------+
         |   flow|hour|vehicle_count|
         +-------+----+-------------+
         | lowest|   2|         1167|
         |highest|  16|        38655|
         +-------+----+-------------+
```

Check in Cassandra.



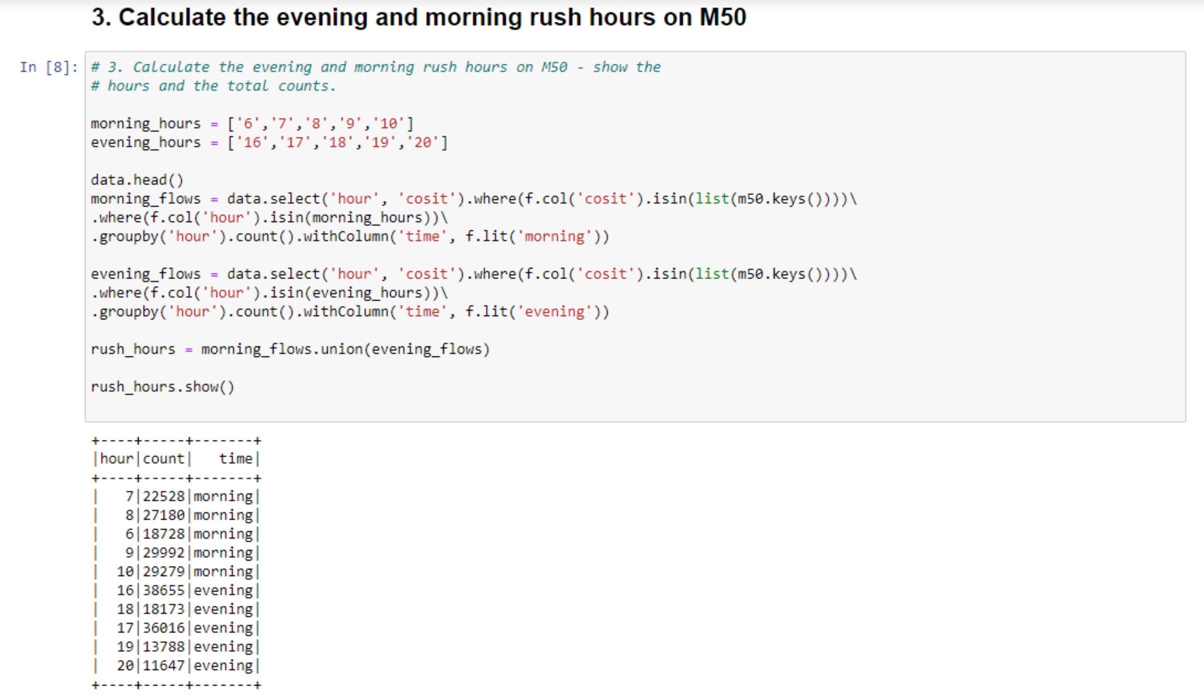
```
cqlsh:sensors> SELECT * FROM motorway_hourly_flows;

 flow     | hour | vehicle_count
---------+------+---------------
  lowest |    2 |          1167
 highest |   16 |         38655

(2 rows)
cqlsh:sensors> _
```

Create Cassandra data structure for motorway rush hours.



```
cqlsh:sensors> CREATE TABLE motorway_rush_hours (
           ... flow text,
           ... hour int,
           ... vehicle_count int,
           ... PRIMARY KEY (flow)
           ... );
```

Perform Computation.

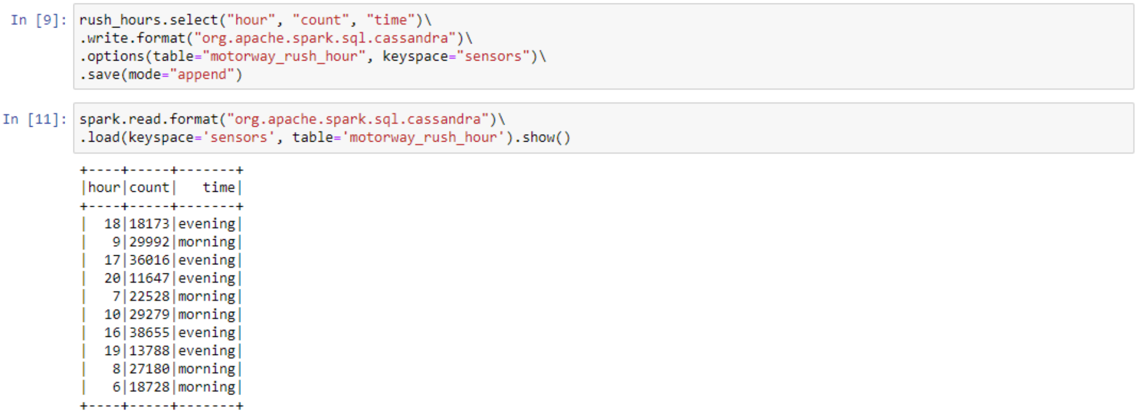


3. Calculate the evening and morning rush hours on M50

```
In [8]: # 3. Calculate the evening and morning rush hours on M50 - show the
        # hours and the total counts.

        morning_hours = ['6','7','8','9','10']
        evening_hours = ['16','17','18','19','20']

        data.head()
        morning_flows = data.select('hour', 'cosit').where(f.col('cosit').isin(list(m50.keys())))\
        .where(f.col('hour').isin(morning_hours))\
        .groupby('hour').count().withColumn('time', f.lit('morning'))

        evening_flows = data.select('hour', 'cosit').where(f.col('cosit').isin(list(m50.keys())))\
        .where(f.col('hour').isin(evening_hours))\
        .groupby('hour').count().withColumn('time', f.lit('evening'))

        rush_hours = morning_flows.union(evening_flows)

        rush_hours.show()
```

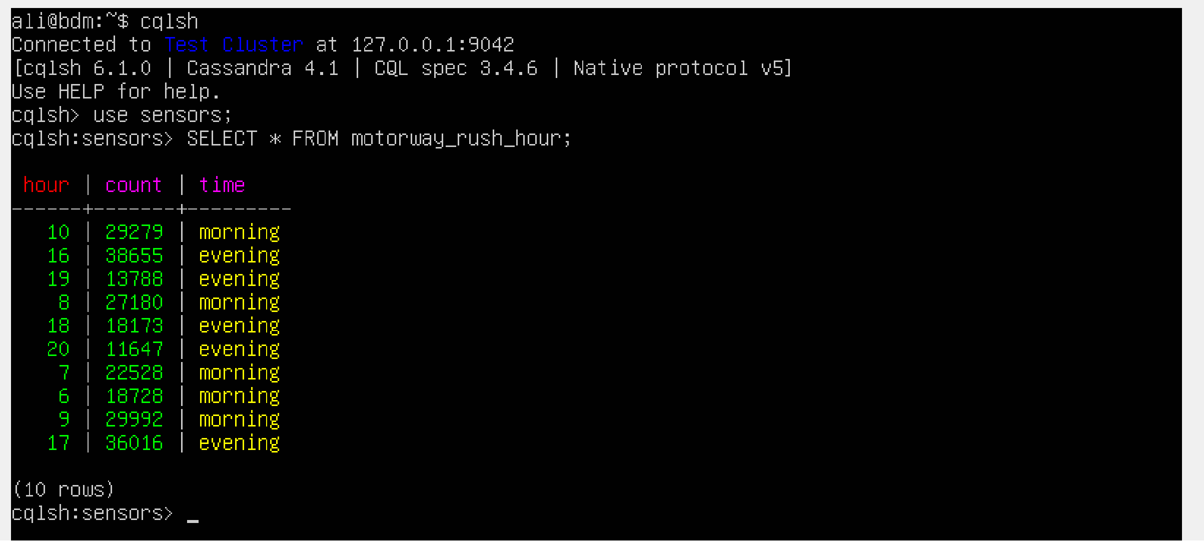
```
+----+-----+-------+
|hour|count|   time|
+----+-----+-------+
|   7|22528|morning|
|   8|27180|morning|
|   6|18728|morning|
|   9|29992|morning|
|  10|29279|morning|
|  16|38655|evening|
|  18|18173|evening|
|  17|36016|evening|
|  19|13788|evening|
|  20|11647|evening|
+----+-----+-------+
```

Move Dataframe to Cassandra and Check.



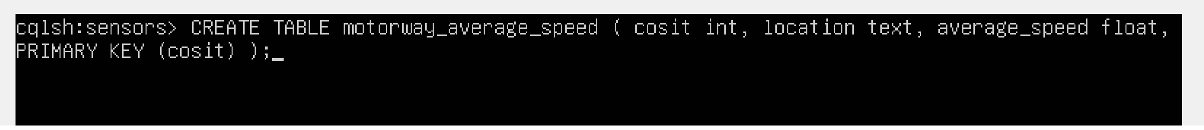
```
In [9]:  rush_hours.select("hour", "count", "time")\
         .write.format("org.apache.spark.sql.cassandra")\
         .options(table="motorway_rush_hour", keyspace="sensors")\
         .save(mode="append")

In [11]: spark.read.format("org.apache.spark.sql.cassandra")\
         .load(keyspace='sensors', table='motorway_rush_hour').show()

         +----+-----+-------+
         |hour|count|   time|
         +----+-----+-------+
         |  18|18173|evening|
         |   9|29992|morning|
         |  17|36016|evening|
         |  20|11647|evening|
         |   7|22528|morning|
         |  10|29279|morning|
         |  16|38655|evening|
         |  19|13788|evening|
         |   8|27180|morning|
         |   6|18728|morning|
         +----+-----+-------+
```

Check in Cassandra.



```
ali@bdm:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.1.0 | Cassandra 4.1 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh> use sensors;
cqlsh:sensors> SELECT * FROM motorway_rush_hour;

 hour | count | time
------+-------+---------
   10 | 29279 | morning
   16 | 38655 | evening
   19 | 13788 | evening
    8 | 27180 | morning
   18 | 18173 | evening
   20 | 11647 | evening
    7 | 22528 | morning
    6 | 18728 | morning
    9 | 29992 | morning
   17 | 36016 | evening

(10 rows)
cqlsh:sensors> _
```

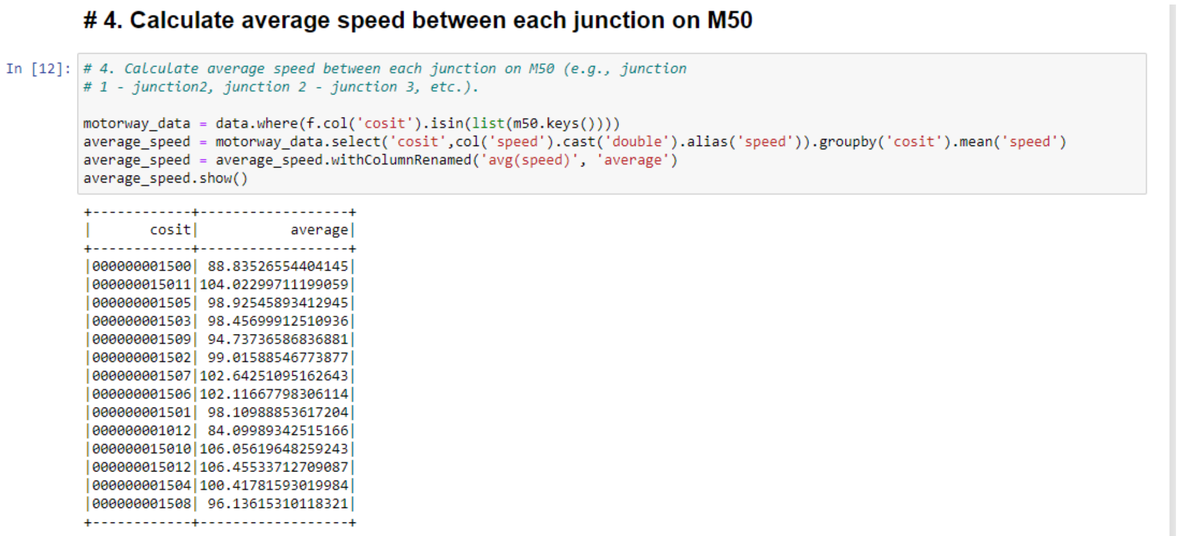
Create a Cassandra data structure for motorway average speed.



```
cqlsh:sensors> CREATE TABLE motorway_average_speed ( cosit int, location text, average_speed float,
PRIMARY KEY (cosit) );_
```

Perform Computation.

First get average speed by cosit.

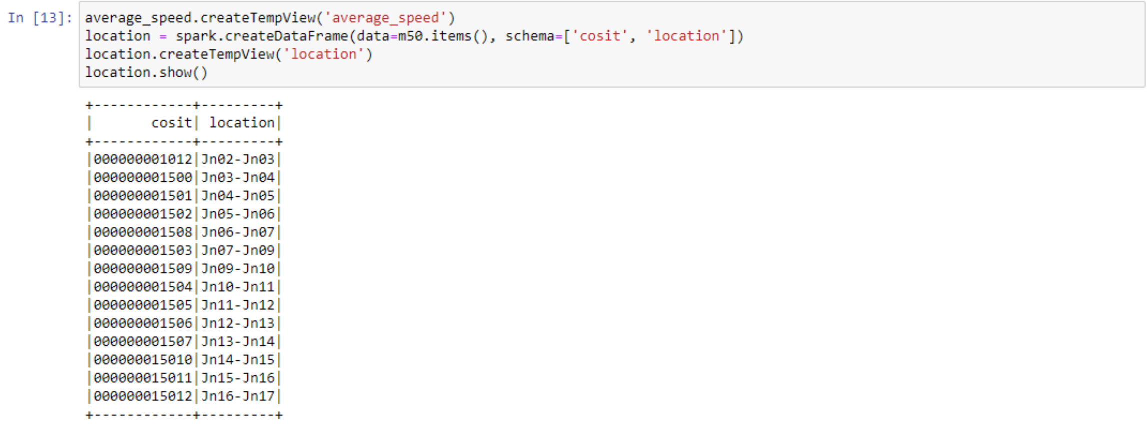


4. Calculate average speed between each junction on M50

```
In [12]: # 4. Calculate average speed between each junction on M50 (e.g., junction
         # 1 - junction2, junction 2 - junction 3, etc.).

         motorway_data = data.where(f.col('cosit').isin(list(m50.keys())))
         average_speed = motorway_data.select('cosit',col('speed').cast('double').alias('speed')).groupby('cosit').mean('speed')
         average_speed = average_speed.withColumnRenamed('avg(speed)', 'average')
         average_speed.show()
```

```
+------------+-----------------+
|       cosit|          average|
+------------+-----------------+
|000000001500| 88.83526554404145|
|000000015011|104.0229971199059|
|000000001505| 98.92545893412945|
|000000001503| 98.45699912510936|
|000000001509| 94.73736586836881|
|000000001502| 99.01588546773877|
|000000001507|102.64251095162643|
|000000001506|102.11667798306114|
|000000001501| 98.10988853617204|
|000000001012| 84.09989342515166|
|000000015010|106.05619648259243|
|000000015012|106.45533712709087|
|000000001504|100.41781593019984|
|000000001508| 96.13615310118321|
+------------+-----------------+
```

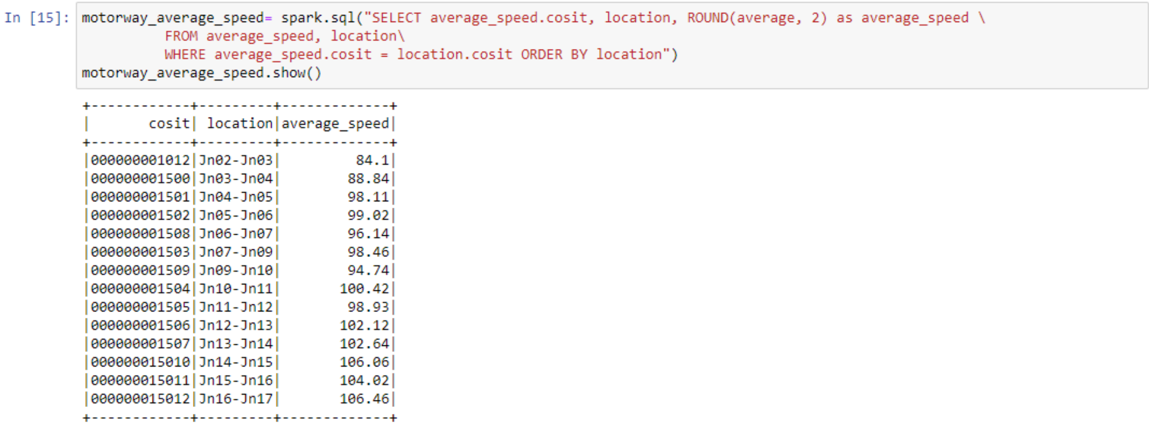
Than get location by cosit.



```
In [13]: average_speed.createTempView('average_speed')
         location = spark.createDataFrame(data=m50.items(), schema=['cosit', 'location'])
         location.createTempView('location')
         location.show()
```

```
+------------+---------+
|       cosit| location|
+------------+---------+
|000000001012|Jn02-Jn03|
|000000001500|Jn03-Jn04|
|000000001501|Jn04-Jn05|
|000000001502|Jn05-Jn06|
|000000001508|Jn06-Jn07|
|000000001503|Jn07-Jn09|
|000000001509|Jn09-Jn10|
|000000001504|Jn10-Jn11|
|000000001505|Jn11-Jn12|
|000000001506|Jn12-Jn13|
|000000001507|Jn13-Jn14|
|000000015010|Jn14-Jn15|
|000000015011|Jn15-Jn16|
|000000015012|Jn16-Jn17|
+------------+---------+
```

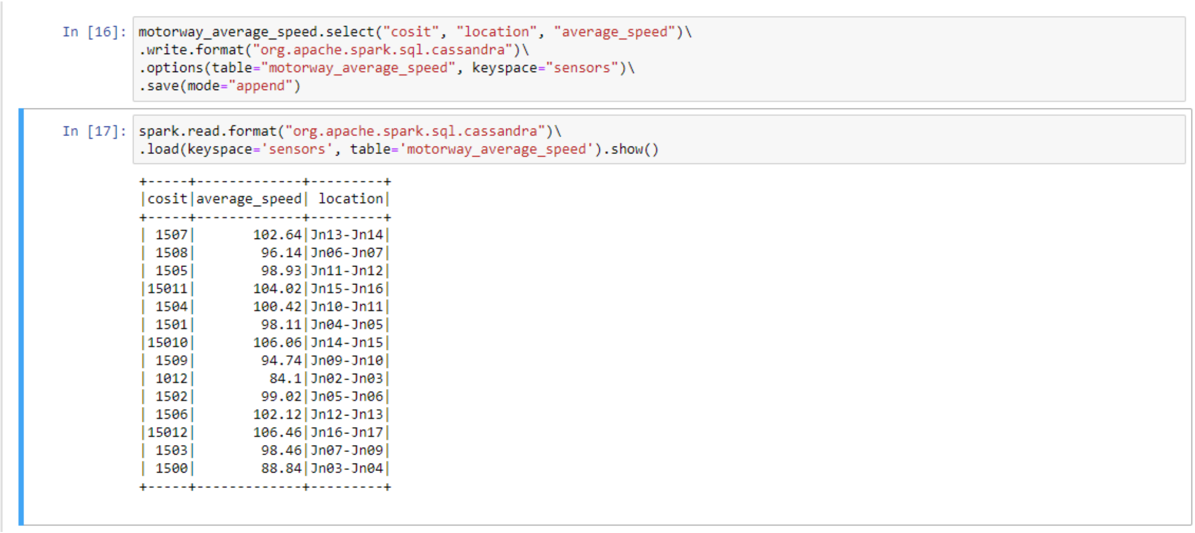
Then joing both tables with cosit as key.



```
In [15]: motorway_average_speed= spark.sql("SELECT average_speed.cosit, location, ROUND(average, 2) as average_speed \
                   FROM average_speed, location\
                   WHERE average_speed.cosit = location.cosit ORDER BY location")
         motorway_average_speed.show()
```

```
+------------+---------+-------------+
|       cosit| location|average_speed|
+------------+---------+-------------+
|000000001012|Jn02-Jn03|         84.1|
|000000001500|Jn03-Jn04|        88.84|
|000000001501|Jn04-Jn05|        98.11|
|000000001502|Jn05-Jn06|        99.02|
|000000001508|Jn06-Jn07|        96.14|
|000000001503|Jn07-Jn09|        98.46|
|000000001509|Jn09-Jn10|        94.74|
|000000001504|Jn10-Jn11|       100.42|
|000000001505|Jn11-Jn12|        98.93|
|000000001506|Jn12-Jn13|       102.12|
|000000001507|Jn13-Jn14|       102.64|
|000000015010|Jn14-Jn15|       106.06|
|000000015011|Jn15-Jn16|       104.02|
|000000015012|Jn16-Jn17|       106.46|
+------------+---------+-------------+
```

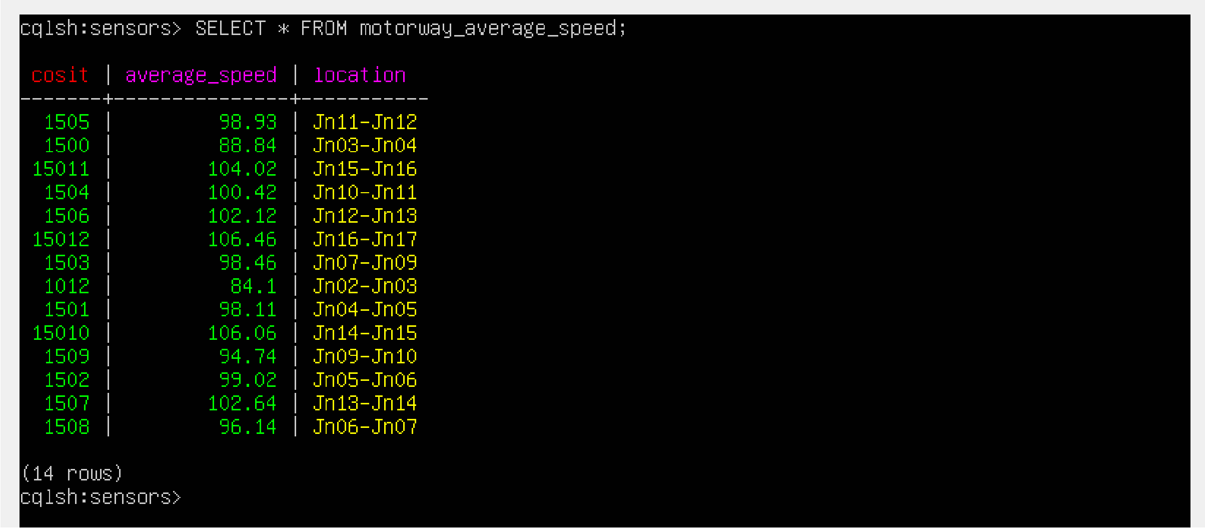
Move to Cassandra and Check.



```
In [16]: motorway_average_speed.select("cosit", "location", "average_speed")\
         .write.format("org.apache.spark.sql.cassandra")\
         .options(table="motorway_average_speed", keyspace="sensors")\
         .save(mode="append")
```

```
In [17]: spark.read.format("org.apache.spark.sql.cassandra")\
         .load(keyspace='sensors', table='motorway_average_speed').show()

         +-----+-------------+---------+
         |cosit|average_speed| location|
         +-----+-------------+---------+
         | 1507|       102.64|Jn13-Jn14|
         | 1508|        96.14|Jn06-Jn07|
         | 1505|        98.93|Jn11-Jn12|
         |15011|       104.02|Jn15-Jn16|
         | 1504|       100.42|Jn10-Jn11|
         | 1501|        98.11|Jn04-Jn05|
         |15010|       106.06|Jn14-Jn15|
         | 1509|        94.74|Jn09-Jn10|
         | 1012|         84.1|Jn02-Jn03|
         | 1502|        99.02|Jn05-Jn06|
         | 1506|       102.12|Jn12-Jn13|
         |15012|       106.46|Jn16-Jn17|
         | 1503|        98.46|Jn07-Jn09|
         | 1500|        88.84|Jn03-Jn04|
         +-----+-------------+---------+
```

Check in Cassandra.



```
cqlsh:sensors> SELECT * FROM motorway_average_speed;

 cosit | average_speed | location
-------+---------------+-----------
  1505 |         98.93 | Jn11-Jn12
  1500 |         88.84 | Jn03-Jn04
 15011 |        104.02 | Jn15-Jn16
  1504 |        100.42 | Jn10-Jn11
  1506 |        102.12 | Jn12-Jn13
 15012 |        106.46 | Jn16-Jn17
  1503 |         98.46 | Jn07-Jn09
  1012 |          84.1 | Jn02-Jn03
  1501 |         98.11 | Jn04-Jn05
 15010 |        106.06 | Jn14-Jn15
  1509 |         94.74 | Jn09-Jn10
  1502 |         99.02 | Jn05-Jn06
  1507 |        102.64 | Jn13-Jn14
  1508 |         96.14 | Jn06-Jn07

(14 rows)
cqlsh:sensors>
```