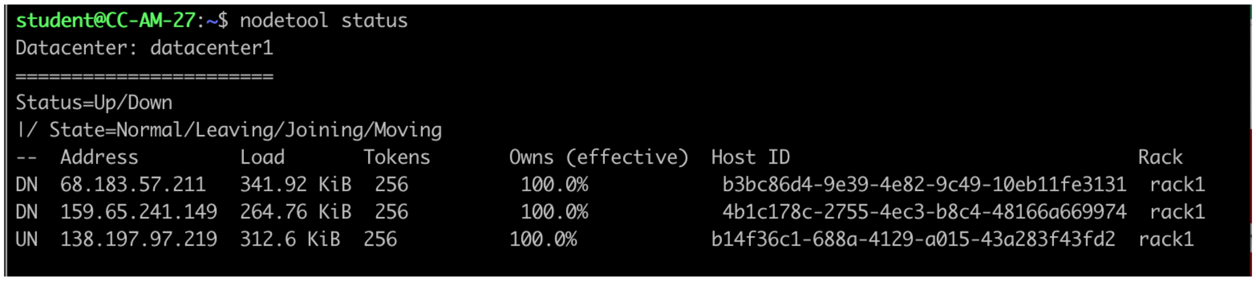
INFSCI 2750: Cloud Computing
Mini Project 3

Yesi Xie (YEX4), Bowen Xu (BOX3), Wenting Wang (WEW77)

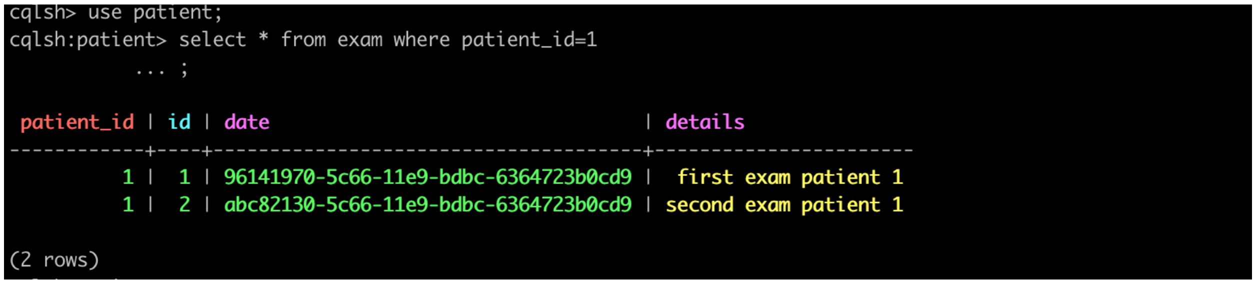
Part1: Configuration

1. Node status:



```
student@CC-AM-27:~$ nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address          Load        Tokens       Owns (effective)  Host ID                               Rack
DN  68.183.57.211    341.92 KiB  256          100.0%            b3bc86d4-9e39-4e82-9c49-10eb11fe3131  rack1
DN  159.65.241.149   264.76 KiB  256          100.0%            4b1c178c-2755-4ec3-b8c4-48166a669974  rack1
UN  138.197.97.219   312.6 KiB   256          100.0%            b14f36c1-688a-4129-a015-43a283f43fd2  rack1
```

2. Test case:

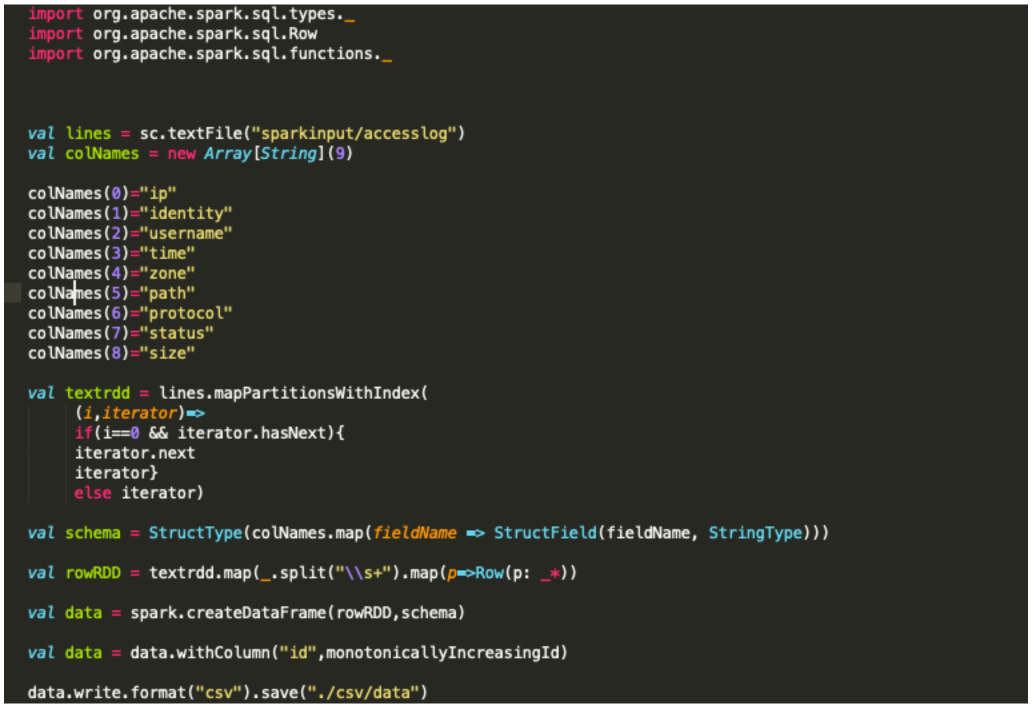


```
cqlsh> use patient;
cqlsh:patient> select * from exam where patient_id=1
           ... ;

 patient_id | id | date                                 | details
------------+----+--------------------------------------+----------------------
          1 |  1 | 96141970-5c66-11e9-bdbc-6364723b0cd9 |  first exam patient 1
          1 |  2 | abc82130-5c66-11e9-bdbc-6364723b0cd9 | second exam patient 1

(2 rows)
```

Part 2: Import Data into Cassandra

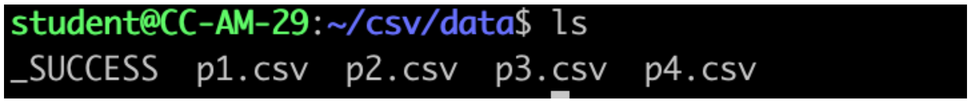
1. Data Pre-processing

- Access_log does not have unique id for each row, which could be used as primary key in table.
- We use Apache Spark to add and id to each row and transfer the access_log file into CSV file.
  Scala Shell:



```
import org.apache.spark.sql.types._
import org.apache.spark.sql.Row
import org.apache.spark.sql.functions._


val lines = sc.textFile("sparkinput/accesslog")
val colNames = new Array[String](9)

colNames(0)="ip"
colNames(1)="identity"
colNames(2)="username"
colNames(3)="time"
colNames(4)="zone"
colNames(5)="path"
colNames(6)="protocol"
colNames(7)="status"
colNames(8)="size"

val textrdd = lines.mapPartitionsWithIndex(
     (i,iterator)=>
     if(i==0 && iterator.hasNext){
     iterator.next
     iterator}
     else iterator)

val schema = StructType(colNames.map(fieldName => StructField(fieldName, StringType)))

val rowRDD = textrdd.map(_.split("\\s+").map(p=>Row(p: _*))

val data = spark.createDataFrame(rowRDD,schema)

val data = data.withColumn("id",monotonicallyIncreasingId)

data.write.format("csv").save("./csv/data")
```

- The CSV file consists of 4 parts



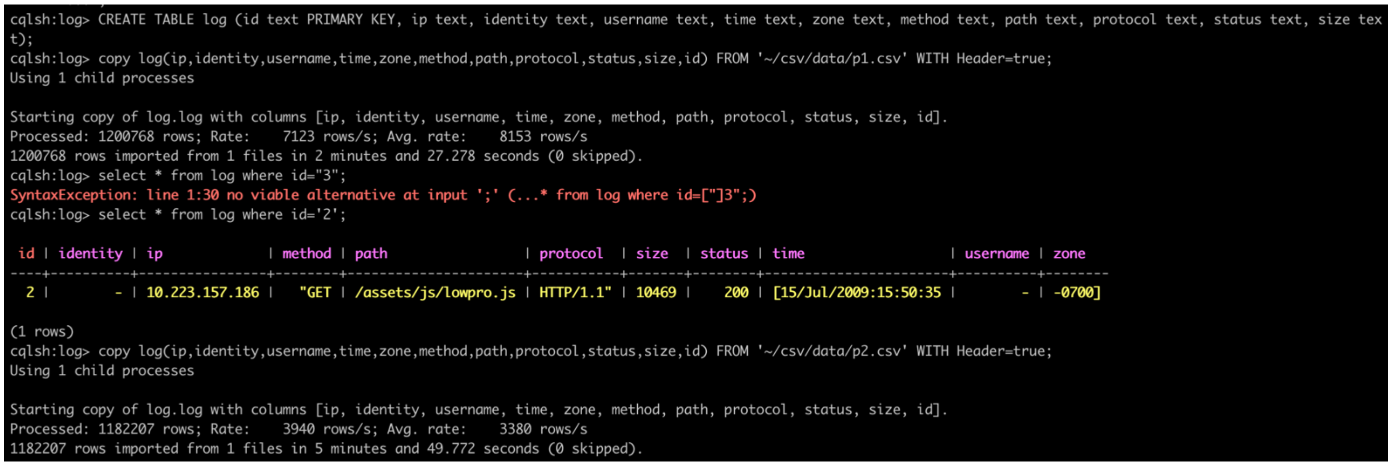
```
student@CC-AM-29:~/csv/data$ ls
_SUCCESS   p1.csv   p2.csv   p3.csv   p4.csv
```

2. Create Keyspace and Table

CREATE KEYSPACE log WITH replication = {'class': 'SimpleStrategy','replication_factor' : 3};

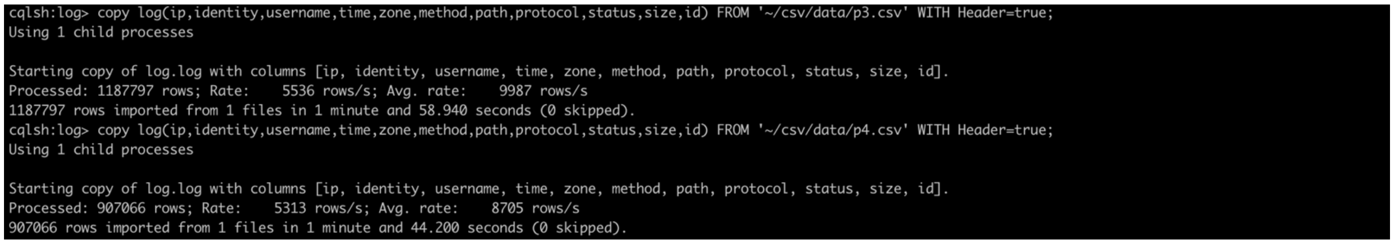
CREATE TABLE log (id int PRIMARY KEY, ip text, identity text, username text, time text, zone text, method text, path text, protocol text, status text, size text);

3. Use copy command to import data from csv into table:



```
cqlsh:log> CREATE TABLE log (id text PRIMARY KEY, ip text, identity text, username text, time text, zone text, method text, path text, protocol text, status text, size tex
t);
cqlsh:log> copy log(ip,identity,username,time,zone,method,path,protocol,status,size,id) FROM '~/csv/data/p1.csv' WITH Header=true;
Using 1 child processes

Starting copy of log.log with columns [ip, identity, username, time, zone, method, path, protocol, status, size, id].
Processed: 1200768 rows; Rate:    7123 rows/s; Avg. rate:    8153 rows/s
1200768 rows imported from 1 files in 2 minutes and 27.278 seconds (0 skipped).
cqlsh:log> select * from log where id="3";
SyntaxException: line 1:30 no viable alternative at input ';' (...* from log where id=["]3";)
cqlsh:log> select * from log where id='2';

 id | identity | ip             | method | path                | protocol  | size  | status | time                  | username | zone
----+----------+----------------+--------+---------------------+-----------+-------+--------+-----------------------+----------+--------
  2 |        - | 10.223.157.186 |   "GET | /assets/js/lowpro.js | HTTP/1.1" | 10469 |    200 | [15/Jul/2009:15:50:35 |        - | -0700]

(1 rows)
cqlsh:log> copy log(ip,identity,username,time,zone,method,path,protocol,status,size,id) FROM '~/csv/data/p2.csv' WITH Header=true;
Using 1 child processes

Starting copy of log.log with columns [ip, identity, username, time, zone, method, path, protocol, status, size, id].
Processed: 1182207 rows; Rate:    3940 rows/s; Avg. rate:    3380 rows/s
1182207 rows imported from 1 files in 5 minutes and 49.772 seconds (0 skipped).
```

An example of row shown above.

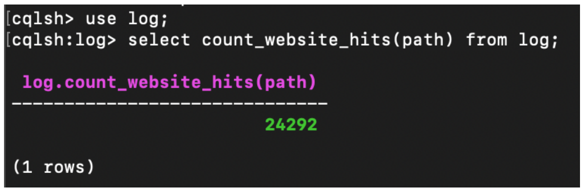


```
cqlsh:log> copy log(ip,identity,username,time,zone,method,path,protocol,status,size,id) FROM '~/csv/data/p3.csv' WITH Header=true;
Using 1 child processes

Starting copy of log.log with columns [ip, identity, username, time, zone, method, path, protocol, status, size, id].
Processed: 1187797 rows; Rate:    5536 rows/s; Avg. rate:    9987 rows/s
1187797 rows imported from 1 files in 1 minute and 58.940 seconds (0 skipped).
cqlsh:log> copy log(ip,identity,username,time,zone,method,path,protocol,status,size,id) FROM '~/csv/data/p4.csv' WITH Header=true;
Using 1 child processes

Starting copy of log.log with columns [ip, identity, username, time, zone, method, path, protocol, status, size, id].
Processed: 907066 rows; Rate:    5313 rows/s; Avg. rate:    8705 rows/s
907066 rows imported from 1 files in 1 minute and 44.200 seconds (0 skipped).
```

Part 3: Operate Data in Cassandra

1. How many hits were made to the website item "/assets/img/release-schedule-logo.png"?

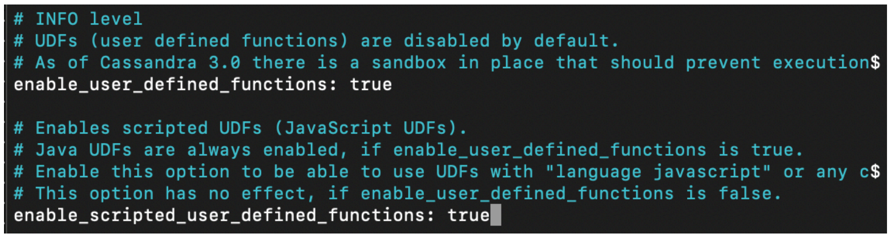
   Answer: 24292 hits.



```
[cqlsh> use log;
[cqlsh:log> select count_website_hits(path) from log;

 log.count_website_hits(path)
------------------------------
                         24292

(1 rows)
```

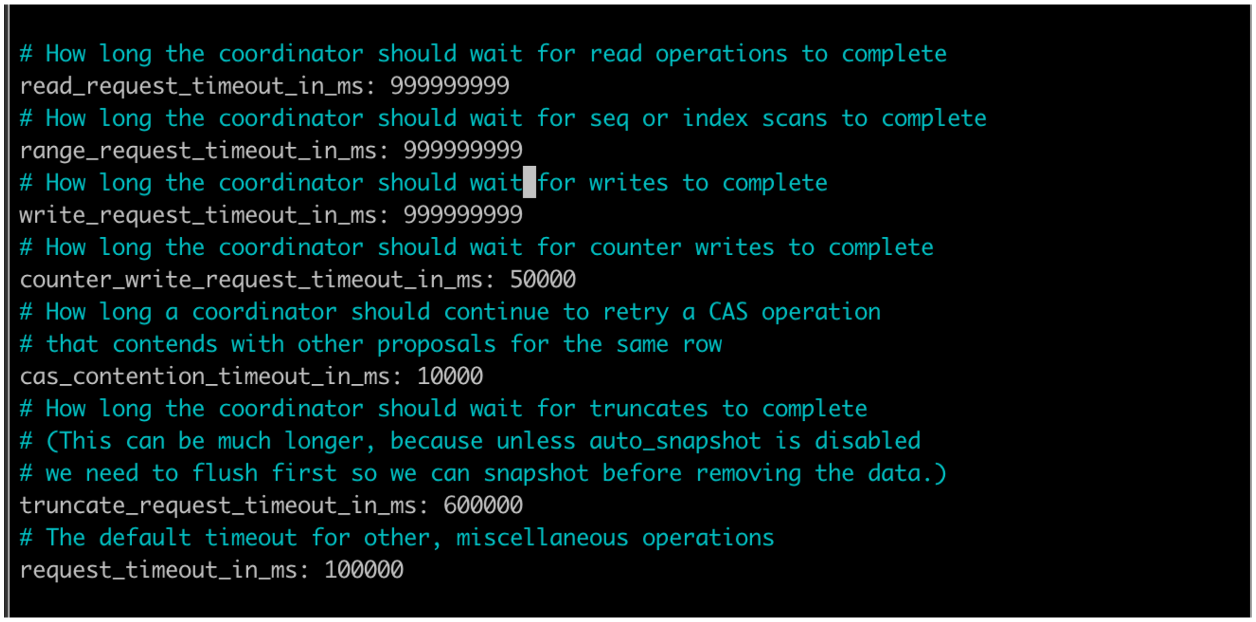
Solution:
Due to the poor performance of executing SELECT and COUNT in a distributes database, we used user-defined function to get the answer.

1> Configuration setting-up:
In Cassandra.yaml, set user-defined function related value to true:



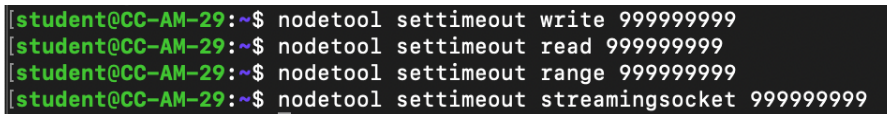
```
# INFO level
# UDFs (user defined functions) are disabled by default.
# As of Cassandra 3.0 there is a sandbox in place that should prevent execution$
enable_user_defined_functions: true

# Enables scripted UDFs (JavaScript UDFs).
# Java UDFs are always enabled, if enable_user_defined_functions is true.
# Enable this option to be able to use UDFs with "language javascript" or any c$
# This option has no effect, if enable_user_defined_functions is false.
enable_scripted_user_defined_functions: true
```

Update Cassandra.yaml for timeout setting:



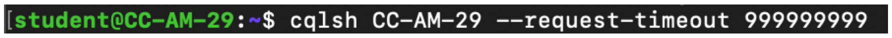
```
# How long the coordinator should wait for read operations to complete
read_request_timeout_in_ms: 999999999
# How long the coordinator should wait for seq or index scans to complete
range_request_timeout_in_ms: 999999999
# How long the coordinator should wait for writes to complete
write_request_timeout_in_ms: 999999999
# How long the coordinator should wait for counter writes to complete
counter_write_request_timeout_in_ms: 50000
# How long a coordinator should continue to retry a CAS operation
# that contends with other proposals for the same row
cas_contention_timeout_in_ms: 10000
# How long the coordinator should wait for truncates to complete
# (This can be much longer, because unless auto_snapshot is disabled
# we need to flush first so we can snapshot before removing the data.)
truncate_request_timeout_in_ms: 600000
# The default timeout for other, miscellaneous operations
request_timeout_in_ms: 100000
```

Alternative approach is manual setting before launch CQL shell:



```
[student@CC-AM-29:~$ nodetool settimeout write 999999999
[student@CC-AM-29:~$ nodetool settimeout read 999999999
[student@CC-AM-29:~$ nodetool settimeout range 999999999
[student@CC-AM-29:~$ nodetool settimeout streamingsocket 999999999
```

2> Run cql shell:



```
[student@CC-AM-29:~$ cqlsh CC-AM-29 --request-timeout 999999999
```

Use log;

3> Run functions(CQL shell):

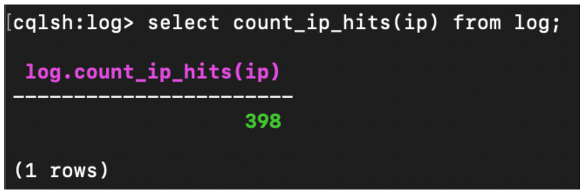
```
CREATE OR REPLACE FUNCTION log_type_count(log map<text, int>, type text)
CALLED ON NULL INPUT
RETURNS map<text, int>
LANGUAGE java AS
' log.put(type, getOrDefault(type, 0)+1);
  return log; ' ;

CREATE OR REPLACE FUNCTION get_website_hits(log map<text, int>)
CALLED ON NULL INPUT
RETURNS int
LANGUAGE java AS
'return (log.containsKey("/assets/img/release-schedule-logo.png"))?
log.get("/assets/img/release-schedule-logo.png"):0;' ;

CREATE OR REPLACE AGGREGATE count_website_hits(text)
SFUNC log_type_count
STYPE map<text,int>
FINALFUNC get_website_hits
INITCOND {};
```

4> Get result:

```
select count_website_hits(path) from log;
```

2. How many hits were made from the IP: 10.207.188.188?

Answer: 398 hits.



```
[cqlsh:log> select count_ip_hits(ip) from log;

 log.count_ip_hits(ip)
-----------------------
                   398

(1 rows)
```

Solution(CQL shell):

1> Run functions:

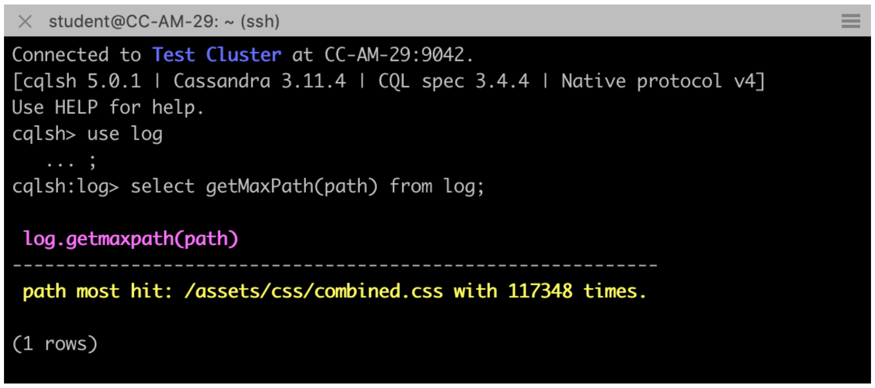
```
CREATE OR REPLACE FUNCTION get_ip_hits(log map<text, int>)
CALLED ON NULL INPUT
RETURNS int
LANGUAGE java AS
'return(log.containsKey("10.207.188.188"))?log.get("10.207.188.188"):0;';

CREATE OR REPLACE AGGREGATE count_ip_hits(text)
SFUNC log_type_count
STYPE map<text,int>
FINALFUNC get_ip_hits
INITCOND {};
```

2> Get result:

```
select count_ip_hits(ip) from log;
```

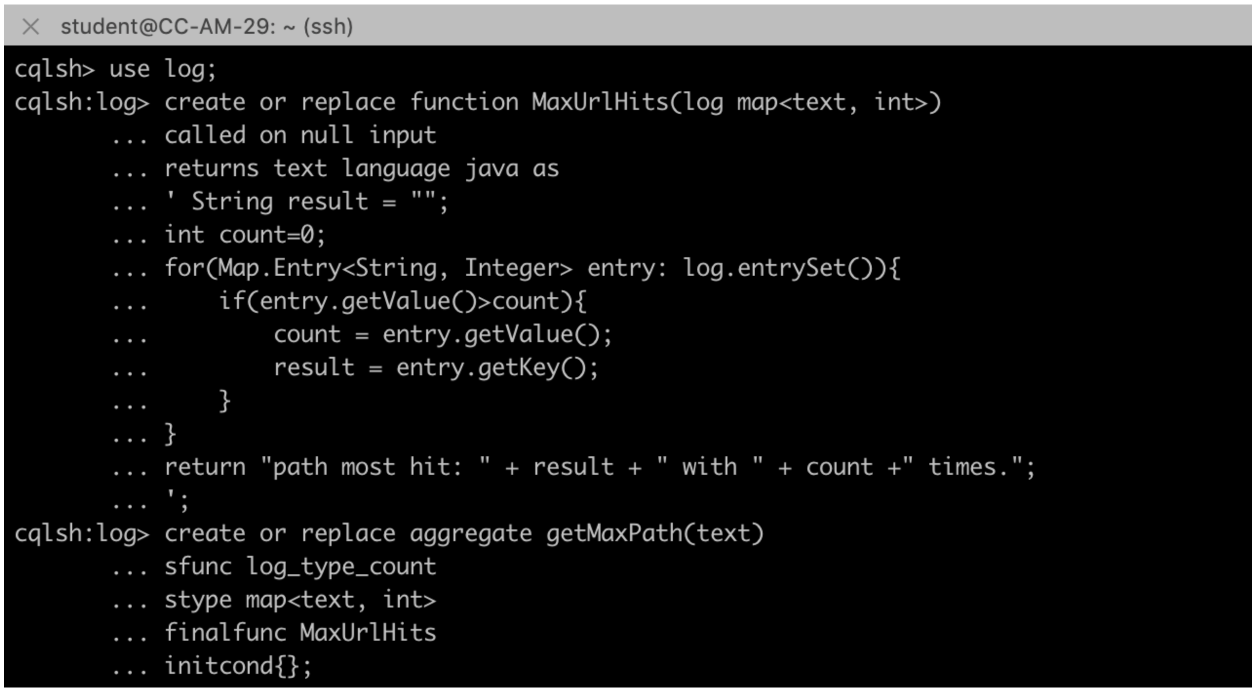
3. Which path in the website has been hit most? How many hits were made to the path?

Answer:



```
Connected to Test Cluster at CC-AM-29:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> use log
   ... ;
cqlsh:log> select getMaxPath(path) from log;

 log.getmaxpath(path)
------------------------------------------------------------
 path most hit: /assets/css/combined.css with 117348 times.

(1 rows)
```

Solution (CQL shell):
1> Run functions:

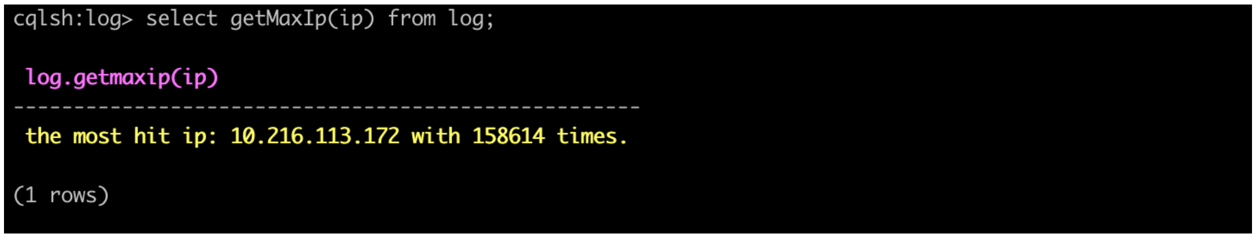


```
cqlsh> use log;
cqlsh:log> create or replace function MaxUrlHits(log map<text, int>)
       ... called on null input
       ... returns text language java as
       ... ' String result = "";
       ... int count=0;
       ... for(Map.Entry<String, Integer> entry: log.entrySet()){
       ...     if(entry.getValue()>count){
       ...         count = entry.getValue();
       ...         result = entry.getKey();
       ...     }
       ... }
       ... return "path most hit: " + result + " with " + count +" times.";
       ... ';
cqlsh:log> create or replace aggregate getMaxPath(text)
       ... sfunc log_type_count
       ... stype map<text, int>
       ... finalfunc MaxUrlHits
       ... initcond{};
```

2> Get result:
select getMaxPath(path) from log;

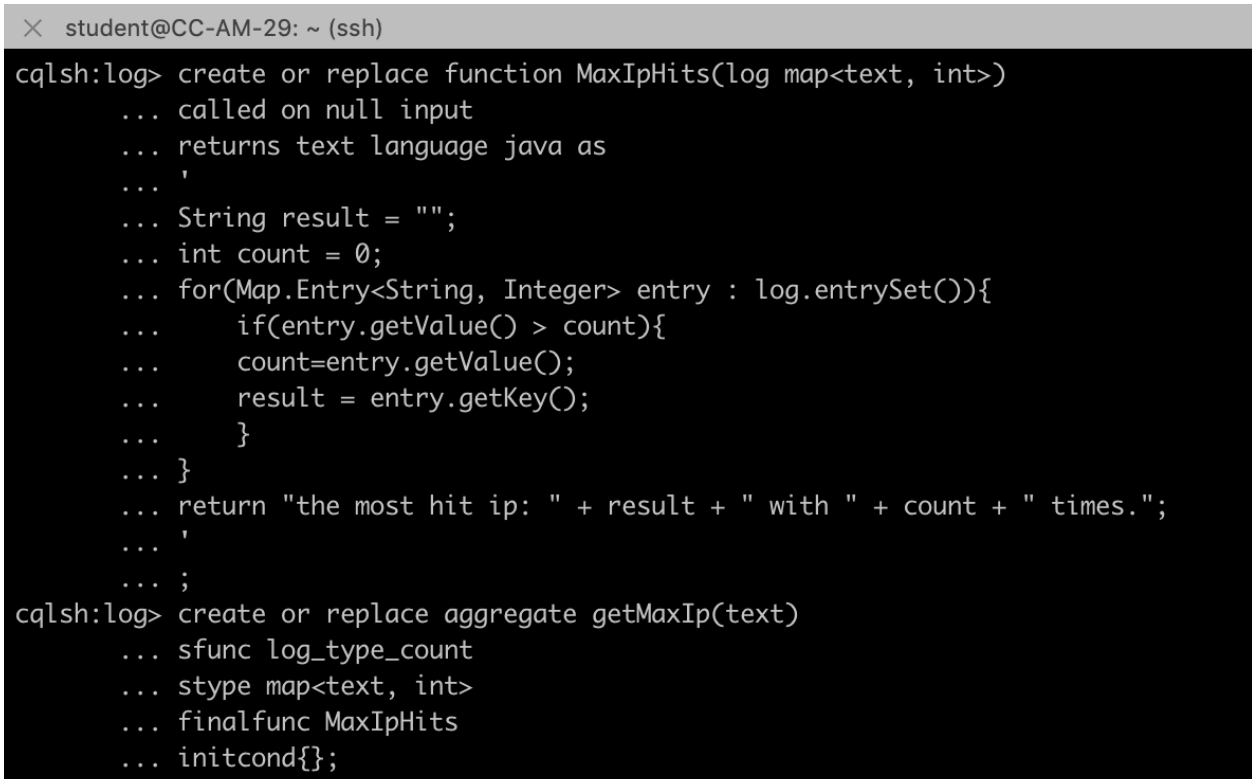
4. **Which IP accesses the website most? How many accesses were made by it?**

Answer:



```
cqlsh:log> select getMaxIp(ip) from log;

 log.getmaxip(ip)
-----------------------------------------------------
 the most hit ip: 10.216.113.172 with 158614 times.

(1 rows)
```

Solution (CQL shell):
1> Run functions:



student@CC-AM-29: ~ (ssh)

```
cqlsh:log> create or replace function MaxIpHits(log map<text, int>)
       ... called on null input
       ... returns text language java as
       ... '
       ... String result = "";
       ... int count = 0;
       ... for(Map.Entry<String, Integer> entry : log.entrySet()){
       ...     if(entry.getValue() > count){
       ...     count=entry.getValue();
       ...     result = entry.getKey();
       ...     }
       ... }
       ... return "the most hit ip: " + result + " with " + count + " times.";
       ... '
       ... ;
cqlsh:log> create or replace aggregate getMaxIp(text)
       ... sfunc log_type_count
       ... stype map<text, int>
       ... finalfunc MaxIpHits
       ... initcond{};
```

2> Get result:
select getMaxIP(ip) from log;