CSCI 5408, Summer 2018

Assignment 1: Search Query Implementation using Relational Database and ElasticSearch

Due: May 24, 2018, 11:59PM

Andrea Christians (B00412494) and Navkaran Kumar (B00782012))

Dalhousie University

Link to GitHub Repository: https://github.com/Navkaran0105/DW_Assgn_1

Table of Contents

Business Scenario

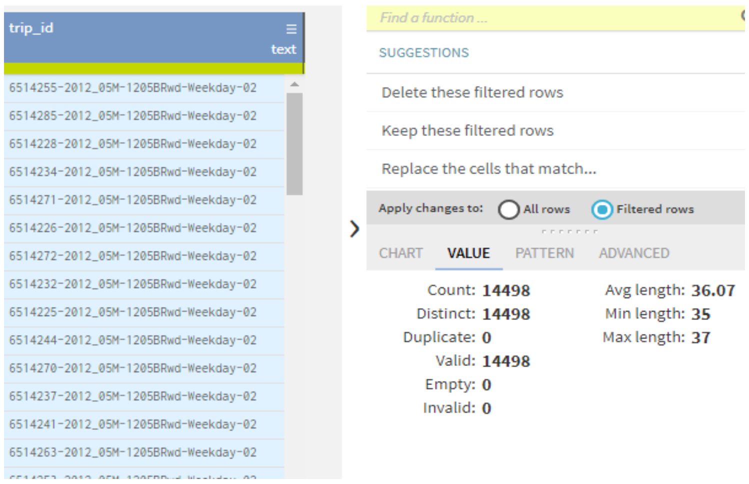
Relational databases as MySQL and NoSQL are widely used in industry and although they can produce similar results, optimization and efficiency is crucial in a business context. This study compares MySQL and ElasticSearch (NoSQL) to determine which method is more efficient. The study will be able to analyze and determine a snapshot of which application is faster in a big data context over four queries where data was retrieved through BrightSpace [4].

Data Analysing and Normalising

Relational Database Management System (RDBMS)

1. For all the transit routes involving *Ferry*, there was a difference in the way *route_id* and *shape_id* was represented in the trips.csv. While importing data into RDBMS, *shape_id* was allocated with a datatype of *INTEGER*, which led to slow data import times and data truncation. Additionally, since the frequency of entries were high, it still appeared to be valid data. Finally, while importing the data from the *trips* table, *shape_id* was given the datatype of *varchar()* to overcome this scenario.

2. While creating relations among tables in MySQL, the length of the column needed to be specified in order to make it a primary or foreign key. In order to determine the lengths of data entries in *trip_id*, Talend Data Preparation Software was used to display the tables and it was found that they did not exceed 50 characters in length (Fig. 1). Thus, the datatype was changed for *trip_id* was changed to *varchar (50)* in both the trips and *stoptimes* table using MySQL Workbench.



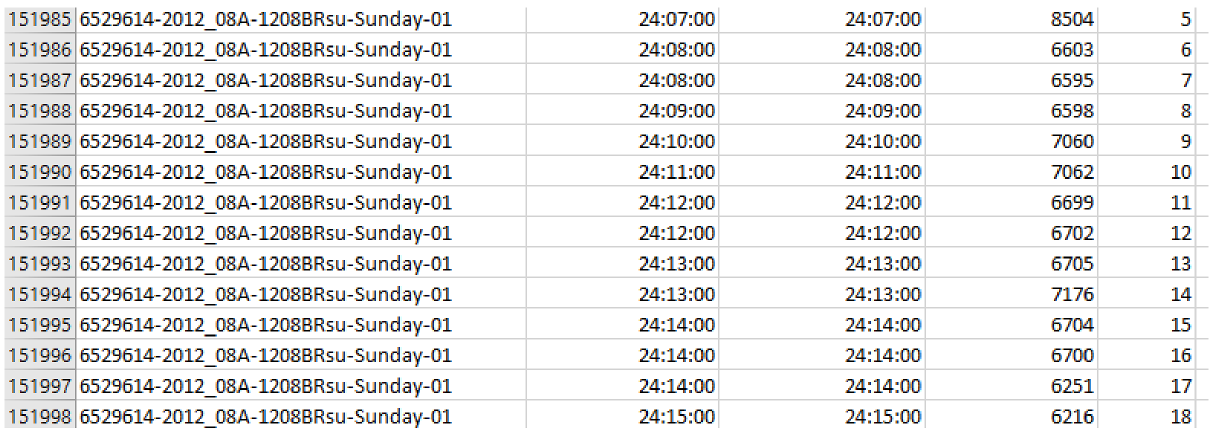
trip_id text
6514255-2012_05M-1205BRwd-Weekday-02
6514285-2012_05M-1205BRwd-Weekday-02
6514228-2012_05M-1205BRwd-Weekday-02
6514234-2012_05M-1205BRwd-Weekday-02
6514271-2012_05M-1205BRwd-Weekday-02
6514226-2012_05M-1205BRwd-Weekday-02
6514272-2012_05M-1205BRwd-Weekday-02
6514232-2012_05M-1205BRwd-Weekday-02
6514225-2012_05M-1205BRwd-Weekday-02
6514244-2012_05M-1205BRwd-Weekday-02
6514270-2012_05M-1205BRwd-Weekday-02
6514237-2012_05M-1205BRwd-Weekday-02
6514241-2012_05M-1205BRwd-Weekday-02
6514263-2012_05M-1205BRwd-Weekday-02

Apply changes to: All rows [✓ Filtered rows]

Count:	14498	Avg length:	36.07
Distinct:	14498	Min length:	35
Duplicate:	0	Max length:	37
Valid:	14498		
Empty:	0		
Invalid:	0		

Figure 1: Screenshot in Talend which depicted the max length of the columns that did not exceed 50 characters.

3. Finally, the *stoptimes* table had arrival and departure time that exceeded 24 hours and were not compliant with the 24-hour format (Fig. 2).



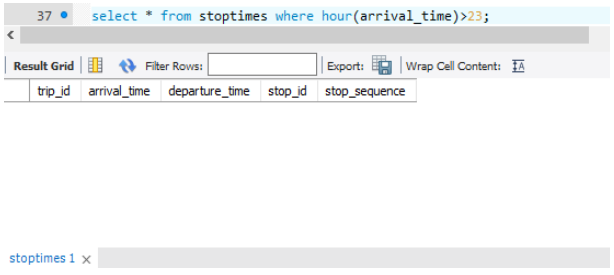
151985	6529614-2012_08A-1208BRsu-Sunday-01	24:07:00	24:07:00	8504	5
151986	6529614-2012_08A-1208BRsu-Sunday-01	24:08:00	24:08:00	6603	6
151987	6529614-2012_08A-1208BRsu-Sunday-01	24:08:00	24:08:00	6595	7
151988	6529614-2012_08A-1208BRsu-Sunday-01	24:09:00	24:09:00	6598	8
151989	6529614-2012_08A-1208BRsu-Sunday-01	24:10:00	24:10:00	7060	9
151990	6529614-2012_08A-1208BRsu-Sunday-01	24:11:00	24:11:00	7062	10
151991	6529614-2012_08A-1208BRsu-Sunday-01	24:12:00	24:12:00	6699	11
151992	6529614-2012_08A-1208BRsu-Sunday-01	24:12:00	24:12:00	6702	12
151993	6529614-2012_08A-1208BRsu-Sunday-01	24:13:00	24:13:00	6705	13
151994	6529614-2012_08A-1208BRsu-Sunday-01	24:13:00	24:13:00	7176	14
151995	6529614-2012_08A-1208BRsu-Sunday-01	24:14:00	24:14:00	6704	15
151996	6529614-2012_08A-1208BRsu-Sunday-01	24:14:00	24:14:00	6700	16
151997	6529614-2012_08A-1208BRsu-Sunday-01	24:14:00	24:14:00	6251	17
151998	6529614-2012_08A-1208BRsu-Sunday-01	24:15:00	24:15:00	6216	18

Figure 1: Sample view of the erroneous times.

The values were replaced and corrected with actual times using a SQL script. Before running the script, the data was imported with a text data type and then it was converted into a '*TIME'* datatype and there was no data truncation. The script was run which updated the values of the arrival and departure time in the *stoptimes* table (see below; Fig. 2). For example, 25:02:50 would be changed to 01:02:50 (that is 1AM).

- update stoptimes set arrival_time=hour(arrival_time)-23 where hour(arrival_time)>23;

- update stoptimes set departure_time=hour(departure_time)-23 where hour(departure_time)>23;



```
select * from stoptimes where hour(arrival_time)>23;
```

trip_id	arrival_time	departure_time	stop_id	stop_sequence

Figure 2: Test query to see if the script worked. As expected, all times past 23:59:59 were corrected.

Data Upload

Uploading data with MySQL Workbench took many hours and so JetBrains DataGrip was used as an alternative tool for uploading data onto to the MySQL Server. The upload speed was faster although altering and querying was preferred on MySQL Workbench due to the minimalistic UI and simplicity.

ElasticSearch

Normalization in ElasticSearch was encountered could have been achieved through two approaches, namely: building a parent-child relationship between the 3 documents and then applying joins, or by restructuring the JSON documents and then applying joins. The former would make the *trips* and *stops* tables as a parent and *stoptimes* as a child of both. However, one child cannot have more than one routing ID so this approach was not applicable to this scenario. The latter involved merging the two documents (*stoptimes* and *stops*) through *stop_id.* The new document would contain all the information from *stoptimes* and *stops* and it is made child with trips as the parent. It was determined that this would be a more viable option and mapping was built around this relation with *trip_id* used as the routing ID.

1. The code for modifying the JSON files was modified from a Ruby script (S. Bhardwaj, personal communications, May 23, 2018). It reads a record from the JSON file and it appends an index and a join entry to each record in the trips document. The script was modified to create a hashmap with *stop_id* as a key and a full record from the *stops* document which was mapped to the key as a value. Both the new index and merged record were written into the new file which was mapped as a child to *trips*.
2. Due to the large size of the merged document, it was subdivided into 11 parts in order to increase the uploading speed considering the RAM in the EC2 instance on AWS.

Data Upload

Data upload and index creation was done through INSOMNIA. The upload wasn't possible for the merged new *stoptimes* document so it was divided into 11 parts and uploaded via INSOMNIA.

Before uploading the data into ElasticSearch, an index was required for each entry. The mapping was pushed using the PUT operation on <EC2 instance ip>:9200/transit (*transit* was the index in this case) and contained all keywords and fields necessary for the join. What mapping basically does is it maps every field to its data type we want to upload data in and then we can query it afterwards using the Elastic Search Queries.

Relational Database

Entity Relationship Diagram

The ERD Diagram relation for the MySQL database (Fig. 3).

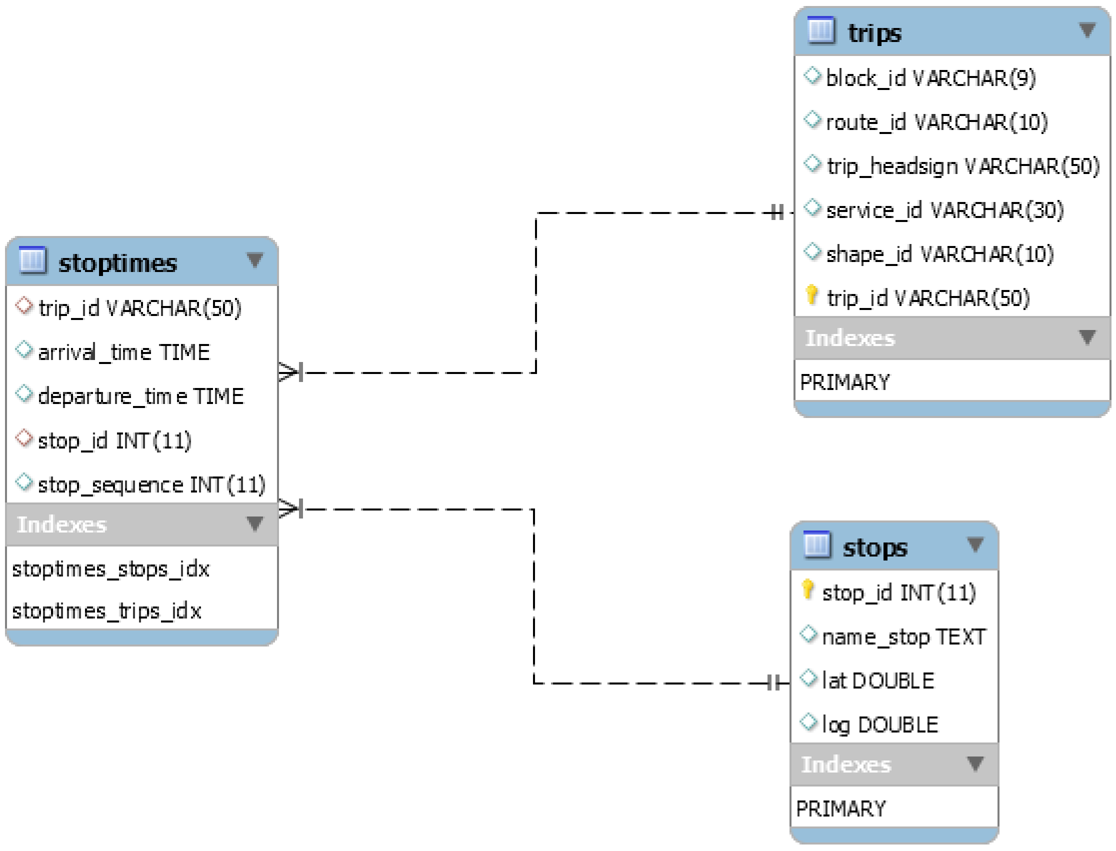


Figure 3: An ERD diagram depicting the joins and schema. The primary key was *stops.stop_id* and *trips.trip_id* and the foreign keys were *stoptimes.trip_id* and *stoptimes.stop_id.*

Application Queries

Performance differences between ElasticSearch and MySQL were compared in the 'Discussion and Summary' section.

MySQL queries

A. **Query:**
```
select distinct(trip_headsign) from trips
join stoptimes s on trips.trip_id = s.trip_id
join stops s2 on s.stop_id = s2.stop_id
        where s2.stop_id =
        (select stop_id from stops where name_stop = "south Park St [northbound]
after South St");
```

B. **Query:**
```
select distinct(trip_headsign) from trips
join stoptimes s on trips.trip_id = s.trip_id
where s.arrival_time between "18:00:00" and "22:00:00";
```

C. **Query:**
```
select trip_id,name_stop,route_id,trip_headsign,arrival_time,departure_time,
stop_sequence,lat,log from trips
join stoptimes using(trip_id)
join stops using(stop_id)
where trip_headsign="330 HALIFAX" and route_id="330-114";
```

D. **Query:**
```
select count(route_id) as BD,name_stop from trips
join stoptimes using(trip_id)
join stops using(stop_id)
        group by stop_id
order by BD desc limit 3;
```

ElasticSearch Queries

A. **Query:**
```
{
 "query": {
  "has_child": {
   "type": "stoptimes",
   "query": {
    "bool": {
     "must": [
                {
                        "match": {
                        "name_stop": "south Park St [northbound] after South St"
                                }
                }
                        ]
    }
   }
  }
 },
        "size":0,
        "aggregations": {
  "distinct": {
```

```
            "terms": {
                "field": "trip_headsign"
            }
        }
                 }
    }
```

B. **Query:**

```
    {
     "query": {
       "has_child": {
         "type": "stoptimes",
         "query": {
           "range" : {
             "arrival_time" : {
                 "gte": "18:00:00",
                 "lte": "22:00:00",
                 "format": "HH:mm:ss"
             }
           }
         }
       }
     },
                "size":0,
                "aggregations": {
       "distinct": {
           "terms": {
               "field": "trip_headsign"
           }
       }
                }
    }
```

C. **Query:**

```
    {
     "query": {
       "has_parent": {
         "parent_type": "trips",
         "query": {
           "bool": {
             "must": [
               {
                 "match": {
                               "trip_headsign": "330 HALIFAX"
                          }
               },
                          {
                               "match": {
                               "route_id": "330-114"
                                   }
                         }

                    ]
    }
   }
  }
 }
}
```

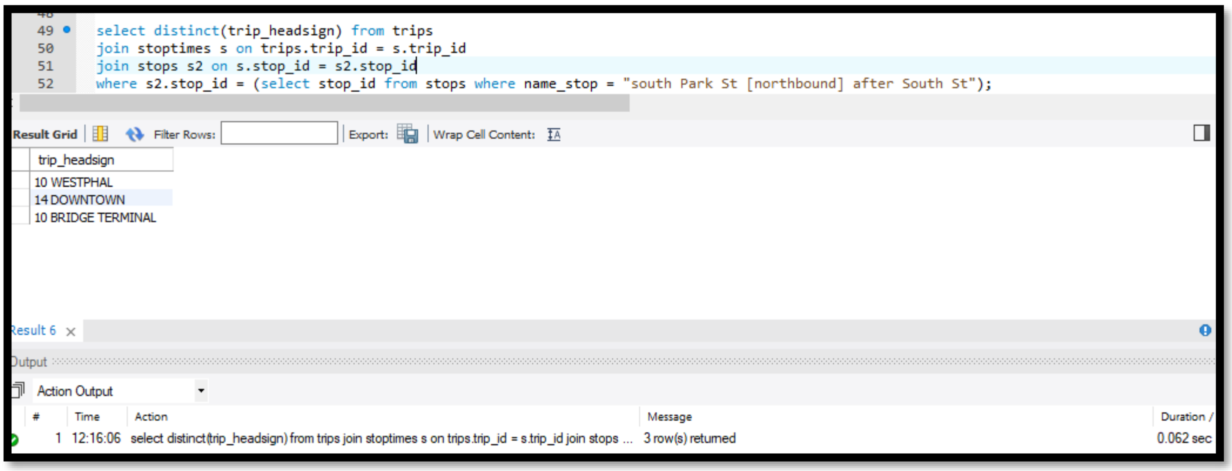
D. **Query :**

```
{
 "size": 0,
      "aggs": {
           "group_by_state": {
                "terms": {
                     "field": "stop_id",
                     "size" : 3
                },
                "aggregations": {
                     "top_three_busy_stops": {
                          "top_hits": {
                               "_source": {
                                         "includes": [
"name_stop" ]
                               },
                               "size" : 1
                          }
                     }
                }
           }
      }
}
```

Test Results

MySQL

A. To find a particular bus that passes through a given stop, the execution of the script starts with a sub-query which retrieves the *stop_id* of the *name_stop*. The *stop_id* then filters out the bus names from the cartesian product of all records. One bus can pass through a bus stop a many times so the keyword *distinct* filters out repeated entries. The 'south Park St [northbound] after South St' bus stop was arbitrarily picked to query (Fig. 4).



```sql
select distinct(trip_headsign) from trips
join stoptimes s on trips.trip_id = s.trip_id
join stops s2 on s.stop_id = s2.stop_id
where s2.stop_id = (select stop_id from stops where name_stop = "south Park St [northbound] after South St");
```

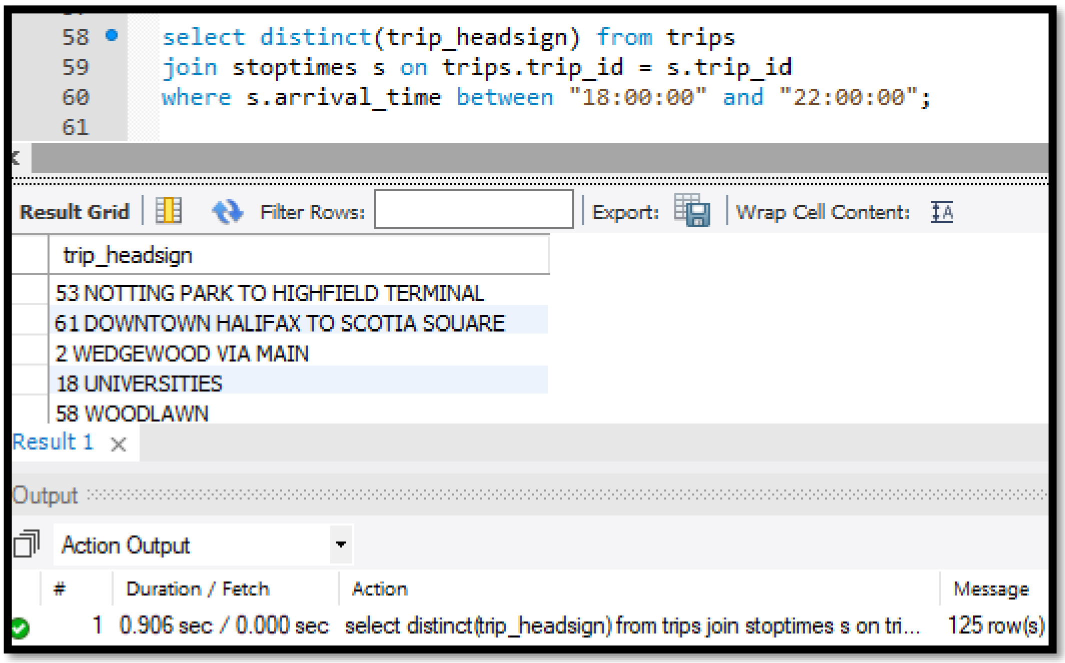
```
trip_headsign
10 WESTPHAL
14 DOWNTOWN
10 BRIDGE TERMINAL
```

```
#  Time      Action                                                                                 Message            Duration /
1  12:16:06  select distinct(trip_headsign) from trips join stoptimes s on trips.trip_id = s.trip_id join stops ...  3 row(s) returned  0.062 sec
```

Figure 4: Data snapshot of the query result for A.

Time: 0.062s

B. To find the buses within a time range, a script that primarily uses two tables, namely *stoptimes* and *trips* using an inner join (Fig. 5). Specifically, bus availability from 18:00:00 to 22:00:00 was queried. Filtering was done with the *where* clause and the *distinct* keyword avoided redundancy.



```sql
select distinct(trip_headsign) from trips
join stoptimes s on trips.trip_id = s.trip_id
where s.arrival_time between "18:00:00" and "22:00:00";
```

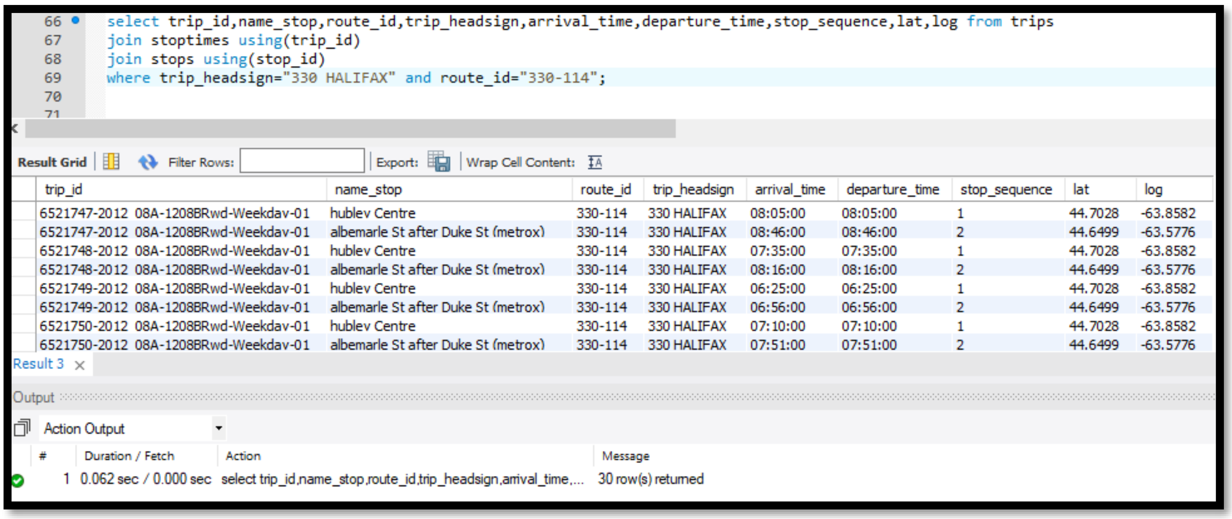
```
trip_headsign
53 NOTTING PARK TO HIGHFIELD TERMINAL
61 DOWNTOWN HALIFAX TO SCOTIA SQUARE
2 WEDGEWOOD VIA MAIN
18 UNIVERSITIES
58 WOODLAWN
```

```
#  Duration / Fetch       Action                                                      Message
1  0.906 sec / 0.000 sec  select distinct(trip_headsign) from trips join stoptimes s on tri...  125 row(s)
```

Figure 5: Data snapshot of the query result for B.

Time: 0.906s

C. The task was retrieve information about a bus on a particular route (Fig. 6). The output gives information about the bus trip from the *trip_id,* the bus stop names from *name_stop*, and arrival and departure times from *arrival_time & departure_time* columns. Additionally, *stop_sequence* columns provide the sequence in which bus stops are arranged on a particular trip. *lat & log* also gives the location of the bus stop.



```
select trip_id,name_stop,route_id,trip_headsign,arrival_time,departure_time,stop_sequence,lat,log from trips
join stoptimes using(trip_id)
join stops using(stop_id)
where trip_headsign="330 HALIFAX" and route_id="330-114";
```

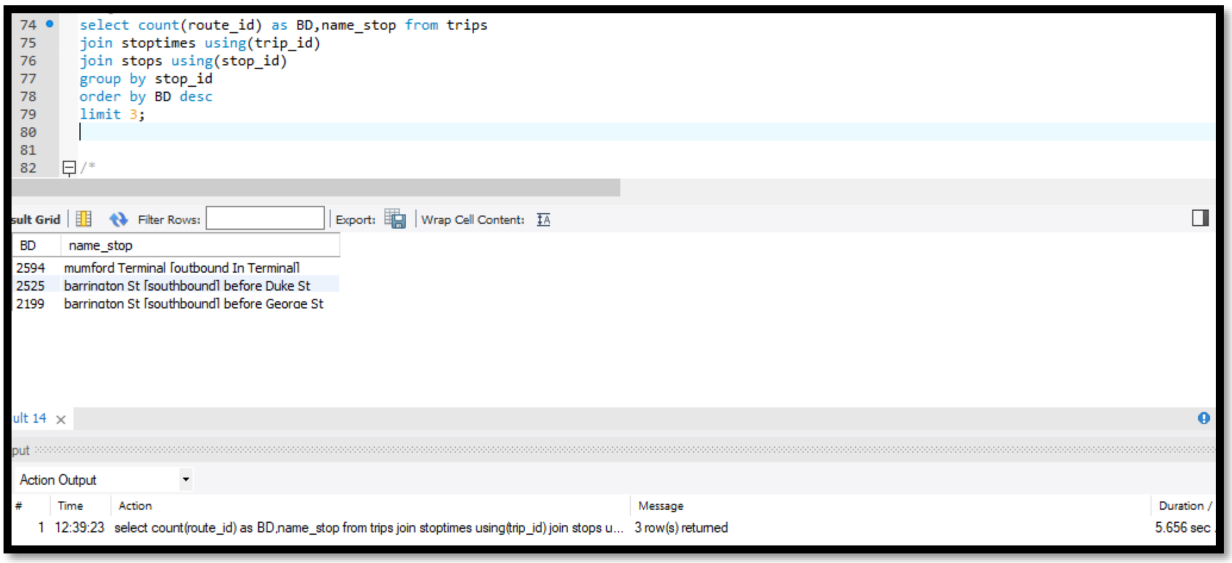
trip_id	name_stop	route_id	trip_headsign	arrival_time	departure_time	stop_sequence	lat	log
6521747-2012 08A-1208BRwd-Weekdav-01	hublev Centre	330-114	330 HALIFAX	08:05:00	08:05:00	1	44.7028	-63.8582
6521747-2012 08A-1208BRwd-Weekdav-01	albemarle St after Duke St (metrox)	330-114	330 HALIFAX	08:46:00	08:46:00	2	44.6499	-63.5776
6521748-2012 08A-1208BRwd-Weekdav-01	hublev Centre	330-114	330 HALIFAX	07:35:00	07:35:00	1	44.7028	-63.8582
6521748-2012 08A-1208BRwd-Weekdav-01	albemarle St after Duke St (metrox)	330-114	330 HALIFAX	08:16:00	08:16:00	2	44.6499	-63.5776
6521749-2012 08A-1208BRwd-Weekdav-01	hublev Centre	330-114	330 HALIFAX	06:25:00	06:25:00	1	44.7028	-63.8582
6521749-2012 08A-1208BRwd-Weekdav-01	albemarle St after Duke St (metrox)	330-114	330 HALIFAX	06:56:00	06:56:00	2	44.6499	-63.5776
6521750-2012 08A-1208BRwd-Weekdav-01	hublev Centre	330-114	330 HALIFAX	07:10:00	07:10:00	1	44.7028	-63.8582
6521750-2012 08A-1208BRwd-Weekdav-01	albemarle St after Duke St (metrox)	330-114	330 HALIFAX	07:51:00	07:51:00	2	44.6499	-63.5776

#	Duration / Fetch	Action	Message
1	0.062 sec / 0.000 sec	select trip_id,name_stop,route_id,trip_headsign,arrival_time,...	30 row(s) returned

Figure 6: Data snapshot of the query result for C.

Time: 0.062s

D. To find the three busiest bus stops in Halifax, the *route_id's* were counted at each stop (Fig. 7) . A join statements amalgamated the data from all three tables in order to count the *route_id 's* and then grouped by *stop_id.* The *group by* clause counted the Route ID's individually for each bus stop. The final result is ordered by the count in descending order and to further extract the first three entries, the keyword *limit* was used.



```
select count(route_id) as BD,name_stop from trips
join stoptimes using(trip_id)
join stops using(stop_id)
group by stop_id
order by BD desc
limit 3;

/*
```

BD	name_stop
2594	mumford Terminal [outbound In Terminal]
2525	barrington St [southbound] before Duke St
2199	barrington St [southbound] before George St

#	Time	Action	Message	Duration /
1	12:39:23	select count(route_id) as BD,name_stop from trips join stoptimes using(trip_id) join stops u...	3 row(s) returned	5.656 sec

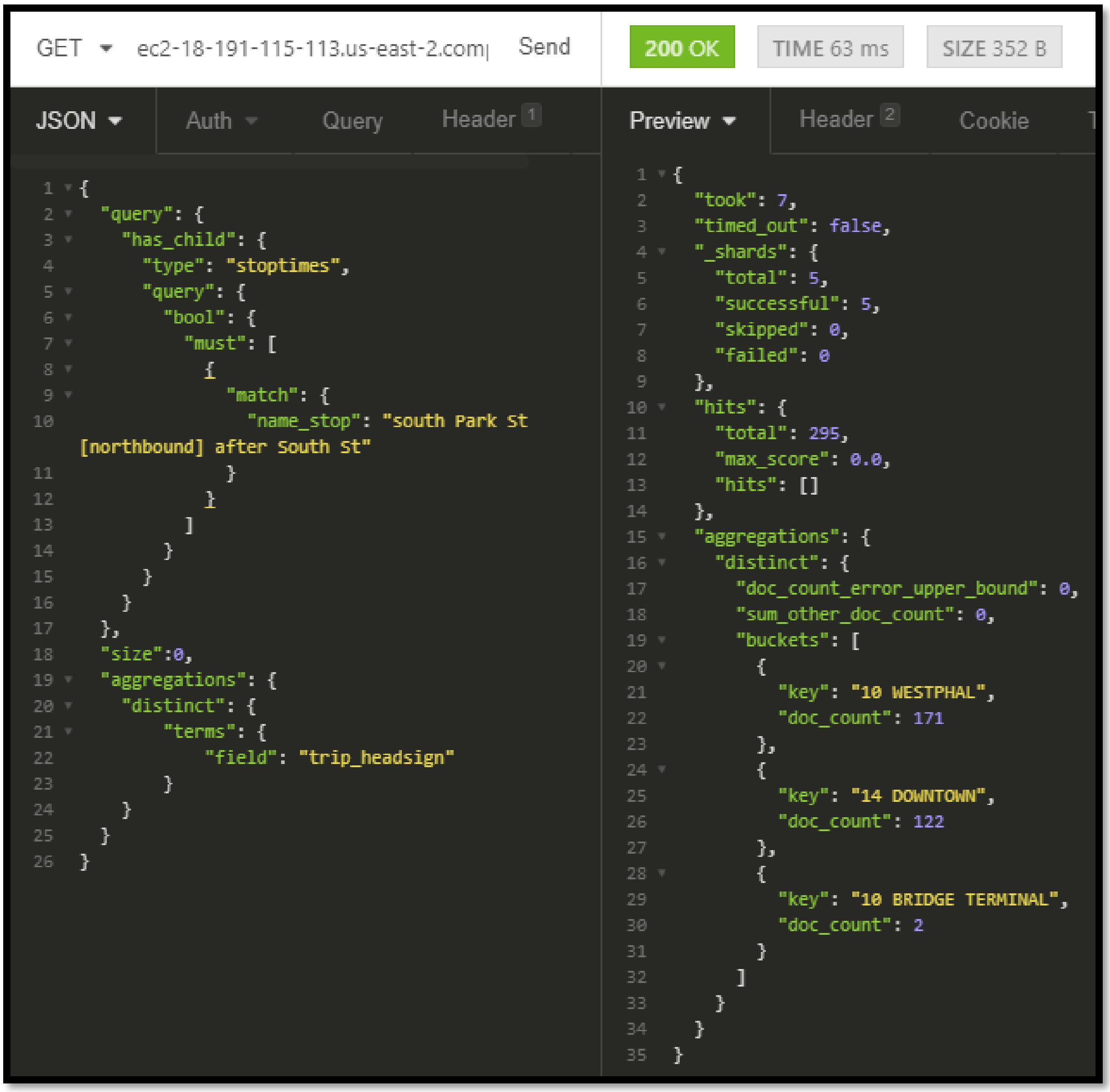
Figure 7: Data snapshot of the query result for D.

Time: 5.656

ElasticSearch

The primary resource for the ElasticSearch queries was the elastic website and the additional tutorial material provided [1], [3].

A. The query was partitioned into 3 parts (Fig. 8). The first section used the parent-child relation between the *trips* and *stoptimes* tables by using the keyword *has_child* and giving it the value of *stoptimes*.
The second part involves *bool* type query, where the *must* keyword to get the names of all the buses that pass from a particular bus stop and the *name_stop* field. The third part is defined by an aggregation property of ES and distinct keyword which filters out the unique *trip_headsign*.

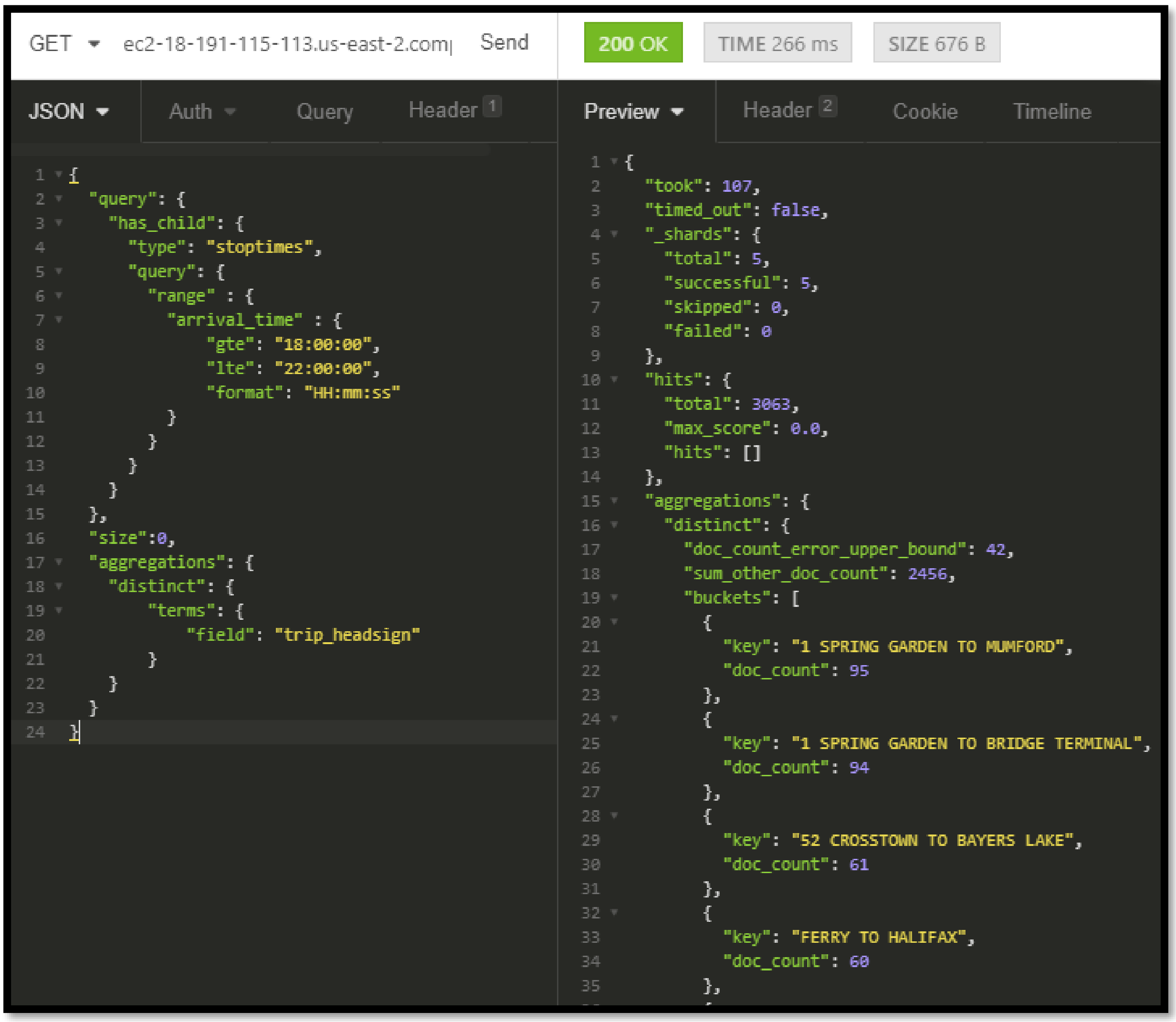


```
{
  "query": {
    "has_child": {
      "type": "stoptimes",
      "query": {
        "bool": {
          "must": [
            {
              "match": {
                "name_stop": "South Park St
[northbound] after South St"
              }
            }
          ]
        }
      }
    }
  },
  "size":0,
  "aggregations": {
    "distinct": {
      "terms": {
        "field": "trip_headsign"
      }
    }
  }
}
```

```
{
  "took": 7,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 295,
    "max_score": 0.0,
    "hits": []
  },
  "aggregations": {
    "distinct": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "10 WESTPHAL",
          "doc_count": 171
        },
        {
          "key": "14 DOWNTOWN",
          "doc_count": 122
        },
        {
          "key": "10 BRIDGE TERMINAL",
          "doc_count": 2
        }
      ]
    }
  }
}
```

Figure 8: Data snapshot of the query result for A.

B. To find the list of buses that ran in a given range, the keyword in ElasticSearch used was *range* (Fig. 9). This required a starting point with a condition (*gte* >> *greater than or equal to*) and an endpoint (*lte* >> *less than or equal to*). The *format* field is optional as it only specifies the format for the time storage and for comparison purposes.

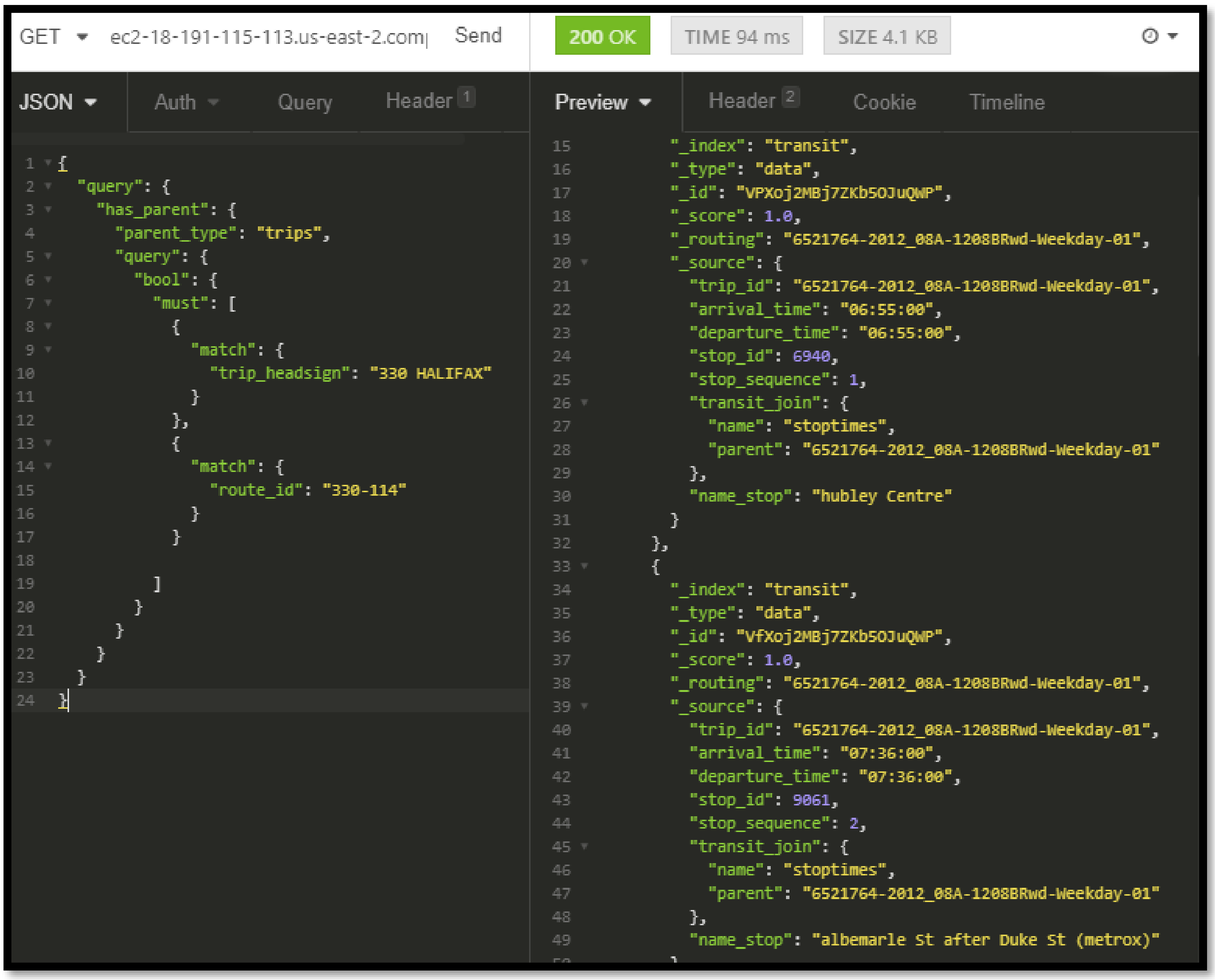


```
{
  "query": {
    "has_child": {
      "type": "stoptimes",
      "query": {
        "range" : {
          "arrival_time" : {
              "gte": "18:00:00",
              "lte": "22:00:00",
              "format": "HH:mm:ss"
          }
        }
      }
    }
  },
  "size":0,
  "aggregations": {
    "distinct": {
        "terms": {
            "field": "trip_headsign"
        }
    }
  }
}
```

```
{
  "took": 107,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 3063,
    "max_score": 0.0,
    "hits": []
  },
  "aggregations": {
    "distinct": {
      "doc_count_error_upper_bound": 42,
      "sum_other_doc_count": 2456,
      "buckets": [
        {
          "key": "1 SPRING GARDEN TO MUMFORD",
          "doc_count": 95
        },
        {
          "key": "1 SPRING GARDEN TO BRIDGE TERMINAL",
          "doc_count": 94
        },
        {
          "key": "52 CROSSTOWN TO BAYERS LAKE",
          "doc_count": 61
        },
        {
          "key": "FERRY TO HALIFAX",
          "doc_count": 60
        },
```

Figure 9: Data snapshot of the query result for B.

C. In order to find the route information of a specific bus, the input is the *trip_headsign* which contains the bus name and the *route_id* (Fig. 10). The *bool* and *must* keywords ensure that both conditions are met for a selection purpose. The input was *330 HALIFAX* for a *trip_headsign* and *330-114* for the *route_id.*
The output retrieves the bus stop names along with the arrival and departure times and the arrival sequence for a particular trip. There may be many buses on the same route so there is variation in stop names and the sequence they arrive in.



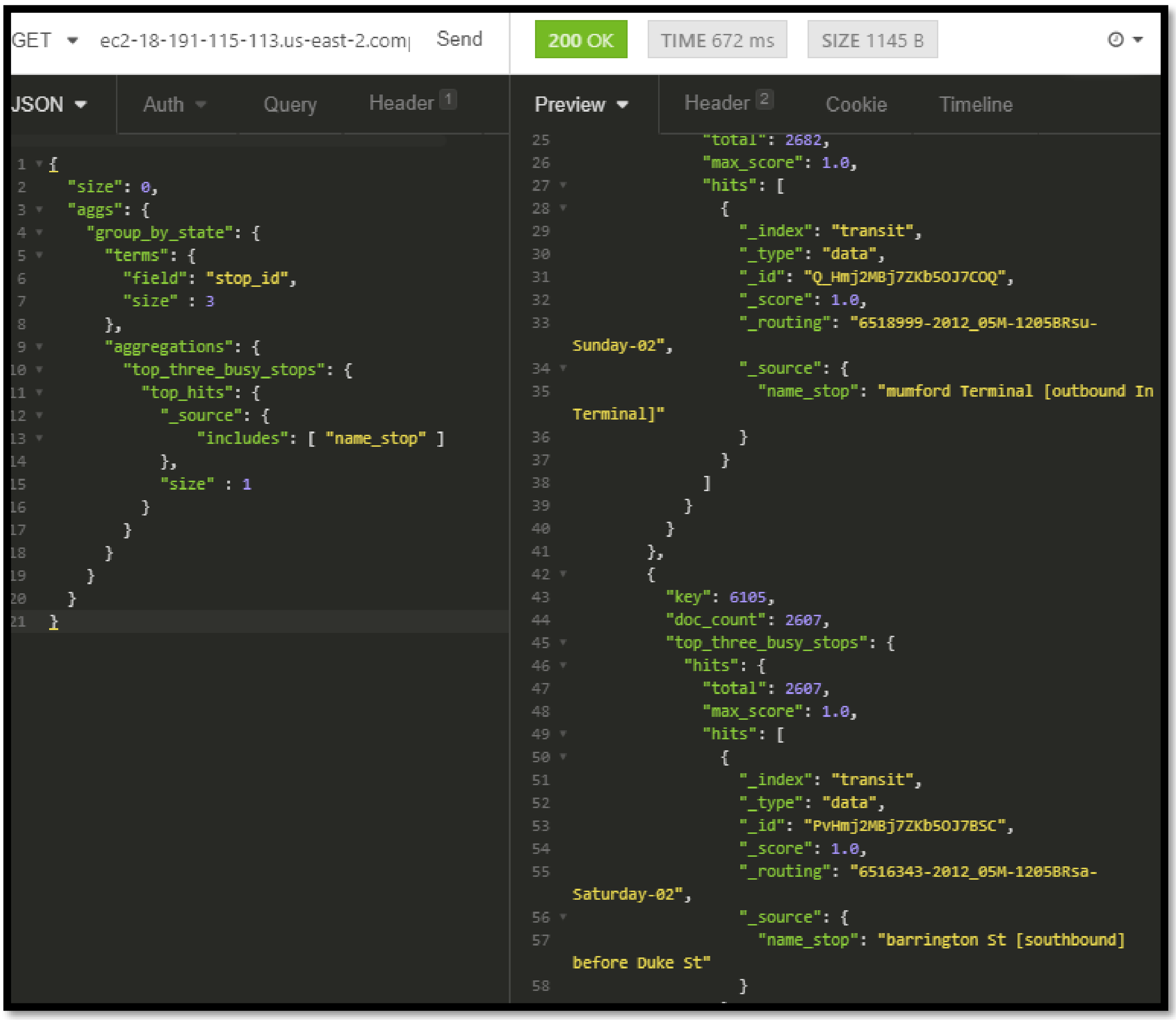
```
{
  "query": {
    "has_parent": {
      "parent_type": "trips",
      "query": {
        "bool": {
          "must": [
            {
              "match": {
                "trip_headsign": "330 HALIFAX"
              }
            },
            {
              "match": {
                "route_id": "330-114"
              }
            }

          ]
        }
      }
    }
  }
}
```

```
          "_index": "transit",
          "_type": "data",
          "_id": "VPXoj2MBj7ZKb5OJuQWP",
          "_score": 1.0,
          "_routing": "6521764-2012_08A-1208BRwd-Weekday-01",
          "_source": {
            "trip_id": "6521764-2012_08A-1208BRwd-Weekday-01",
            "arrival_time": "06:55:00",
            "departure_time": "06:55:00",
            "stop_id": 6940,
            "stop_sequence": 1,
            "transit_join": {
              "name": "stoptimes",
              "parent": "6521764-2012_08A-1208BRwd-Weekday-01"
            },
            "name_stop": "hubley Centre"
          }
        },
        {
          "_index": "transit",
          "_type": "data",
          "_id": "VfXoj2MBj7ZKb5OJuQWP",
          "_score": 1.0,
          "_routing": "6521764-2012_08A-1208BRwd-Weekday-01",
          "_source": {
            "trip_id": "6521764-2012_08A-1208BRwd-Weekday-01",
            "arrival_time": "07:36:00",
            "departure_time": "07:36:00",
            "stop_id": 9061,
            "stop_sequence": 2,
            "transit_join": {
              "name": "stoptimes",
              "parent": "6521764-2012_08A-1208BRwd-Weekday-01"
            },
            "name_stop": "albemarle St after Duke St (metrox)"
```

Figure 10: Data snapshot of the query result for C.

D. To find the busiest three bus stops in Halifax, the *group_by_state* keyword was used with the field keyword in the query on which the result should be grouped on, in this case, *stop_id* (Fig. 11). Despite getting the whole data about bus stops out, it's filtered out by the *includes* keyword putting *name_stop* in the output.



```
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "stop_id",
        "size" : 3
      },
      "aggregations": {
        "top_three_busy_stops": {
          "top_hits": {
            "_source": {
                "includes": [ "name_stop" ]
            },
            "size" : 1
          }
        }
      }
    }
  }
}
```

```
              "total": 2682,
              "max_score": 1.0,
              "hits": [
                {
                  "_index": "transit",
                  "_type": "data",
                  "_id": "Q_Hmj2MBj7ZKb5OJ7COQ",
                  "_score": 1.0,
                  "_routing": "6518999-2012_05M-1205BRsu-
Sunday-02",
                  "_source": {
                    "name_stop": "mumford Terminal [outbound In
Terminal]"
                  }
                }
              ]
            }
          }
        },
        {
          "key": 6105,
          "doc_count": 2607,
          "top_three_busy_stops": {
            "hits": {
              "total": 2607,
              "max_score": 1.0,
              "hits": [
                {
                  "_index": "transit",
                  "_type": "data",
                  "_id": "PvHmj2MBj7ZKb5OJ7BSC",
                  "_score": 1.0,
                  "_routing": "6516343-2012_05M-1205BRsa-
Saturday-02",
                  "_source": {
                    "name_stop": "barrington St [southbound]
before Duke St"
                  }
```

Figure 11: Data snapshot of the query result for A.

Discussion and Summary

In this assignment an instance was created to query two different database types, MySQL and ElasticSearch (NoSQL) to compare the results. The results for the MySQL database were slightly shorter for smaller queries but took longer for larger data requests (Fig. X). Additionally, the queries were slightly more length and less intuitive in text (Fig. 12). In contrast, the ElasticSearch results returned faster results for larger dataset searches (that contained more joins in RDBMS) and the scripts were generally keyword based. For a potential user and developer in Big Data, ElasticSearch returned faster results with less effort. Overall, it appears to be a more powerful tool than MySQL in this project.

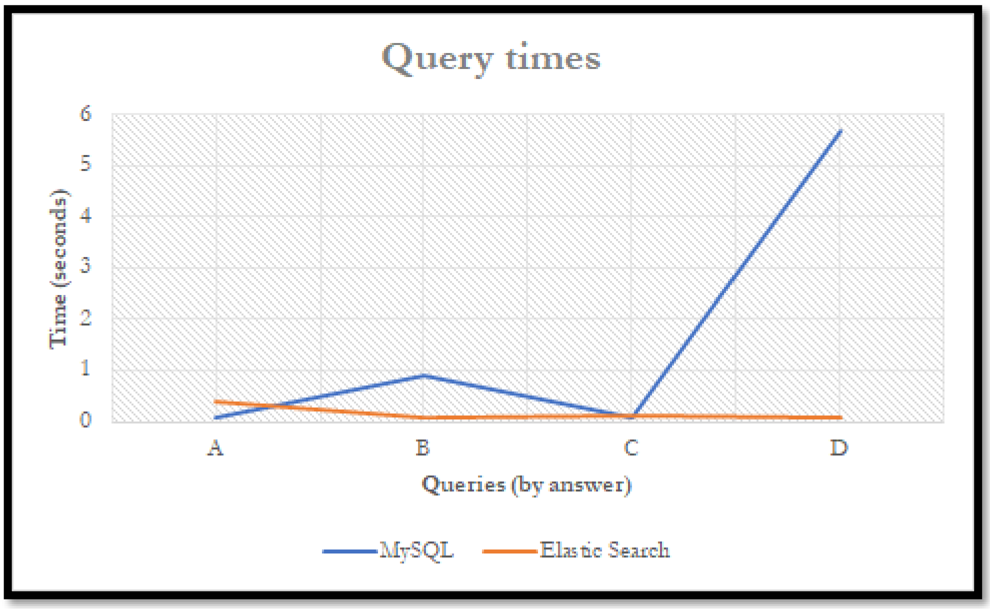


Figure 12: Comparison time between elastic time queries and MySQL. ElasticSearch queries generally returned low query times for all results whereas MySQL varied significantly with larger data query requests.

In terms of software and data processing, using MySQL was significantly easier to normalize and join tables. If this were a large, long-term project querying big data, ElasticSearch may be more useful since the queries are faster, however there were multiple issues with the mapping and indexing of the data and this proved timely. Once the data had been mapped, there was no way to change the mapping after it had been uploaded. Furthermore, altering the JSON files to add indexes was also a timely process and because they were significantly larger in size, they had to be subdivided to be uploaded onto the server. MySQL was therefore the preferred method to work with even though queries were slightly longer.

References

[1]"ElasticSearch Reference [6.x] | Elastic", *Elastic.co*, 2018. [Online]. Available: https://www.elastic.co/guide/en/ElasticSearch/reference/6.x/index.html. [Accessed: 24- May- 2018].

[2]"How to index a .JSON file", *Discuss the Elastic Stack*, 2018. [Online]. Available: https://discuss.elastic.co/t/how-to-index-a-json-file/11539. [Accessed: 24- May- 2018].

[3]"Tutorial 1", *Web.cs.dal.ca*, 2018. [Online]. Available: https://web.cs.dal.ca/~kosmajac/CSCI5408_tutorials/a1.html. [Accessed: 24- May- 2018].

[4]*Dal.brightspace.com*, 2018. [Online]. Available: https://dal.brightspace.com/d2l/le/content/73925/Home. [Accessed: 24- May- 2018].