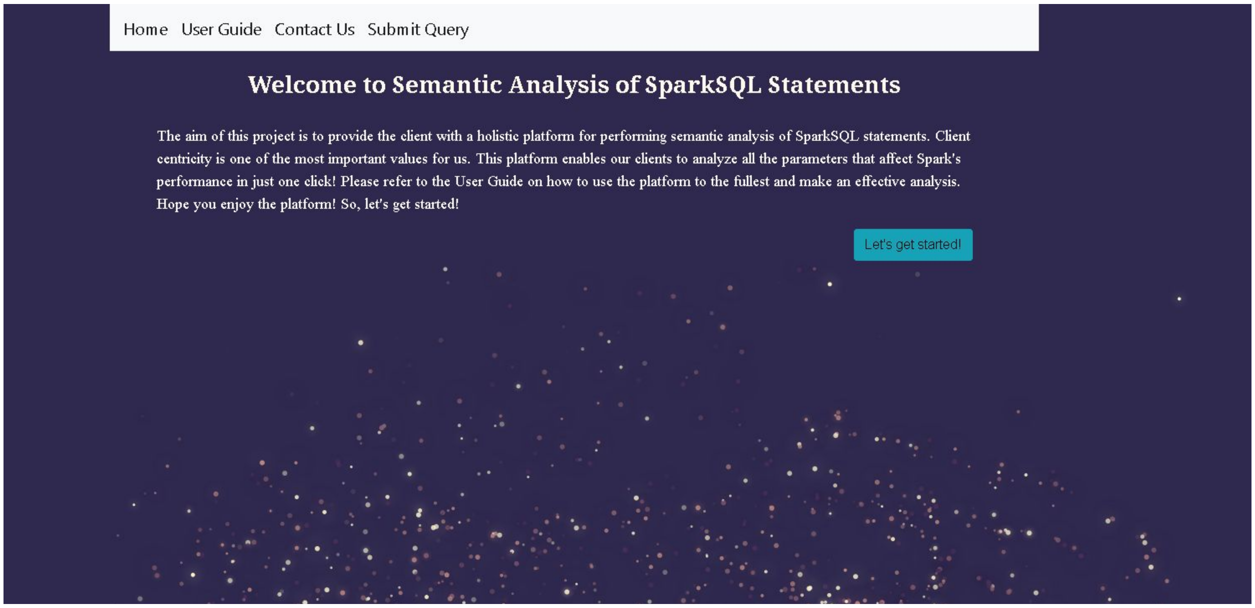
Semantic Analysis of Spark Sql Statements Project UI

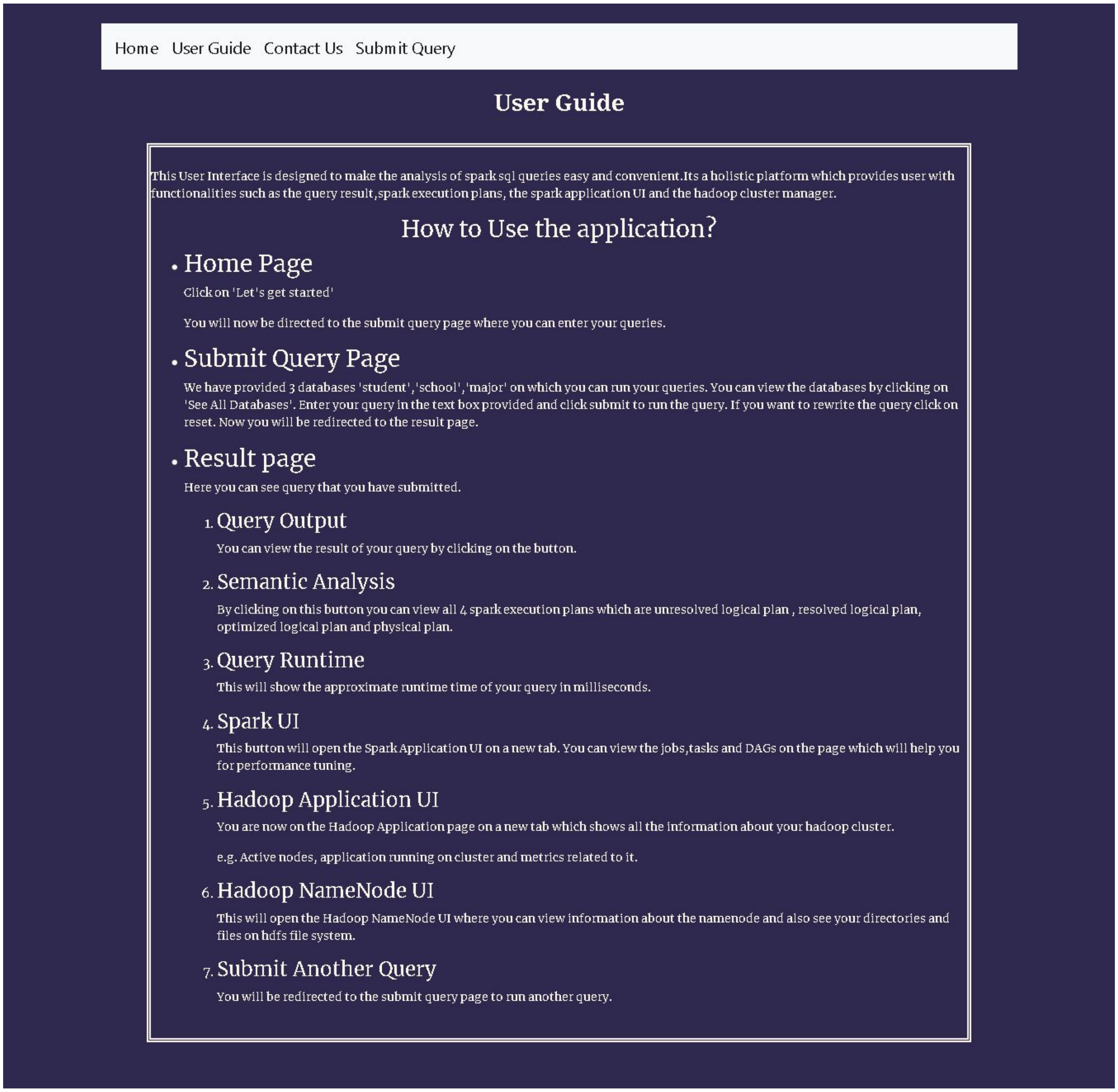
1. Homepage



Welcome to Semantic Analysis of SparkSQL Statements

The aim of this project is to provide the client with a holistic platform for performing semantic analysis of SparkSQL statements. Client centricity is one of the most important values for us. This platform enables our clients to analyze all the parameters that affect Spark's performance in just one click! Please refer to the User Guide on how to use the platform to the fullest and make an effective analysis. Hope you enjoy the platform! So, let's get started!

2. User Guide



User Guide

This User Interface is designed to make the analysis of spark sql queries easy and convenient.Its a holistic platform which provides user with functionalities such as the query result,spark execution plans, the spark application UI and the hadoop cluster manager.

How to Use the application?

- **Home Page**

  Click on 'Let's get started'

  You will now be directed to the submit query page where you can enter your queries.

- **Submit Query Page**

  We have provided 3 databases 'student','school','major' on which you can run your queries. You can view the databases by clicking on 'See All Databases'. Enter your query in the text box provided and click submit to run the query. If you want to rewrite the query click on reset. Now you will be redirected to the result page.

- **Result page**

  Here you can see query that you have submitted.

  1. **Query Output**

     You can view the result of your query by clicking on the button.

  2. **Semantic Analysis**

     By clicking on this button you can view all 4 spark execution plans which are unresolved logical plan , resolved logical plan, optimized logical plan and physical plan.

  3. **Query Runtime**

     This will show the approximate runtime time of your query in milliseconds.

  4. **Spark UI**

     This button will open the Spark Application UI on a new tab. You can view the jobs,tasks and DAGs on the page which will help you for performance tuning.

  5. **Hadoop Application UI**

     You are now on the Hadoop Application page on a new tab which shows all the information about your hadoop cluster.

     e.g. Active nodes, application running on cluster and metrics related to it.

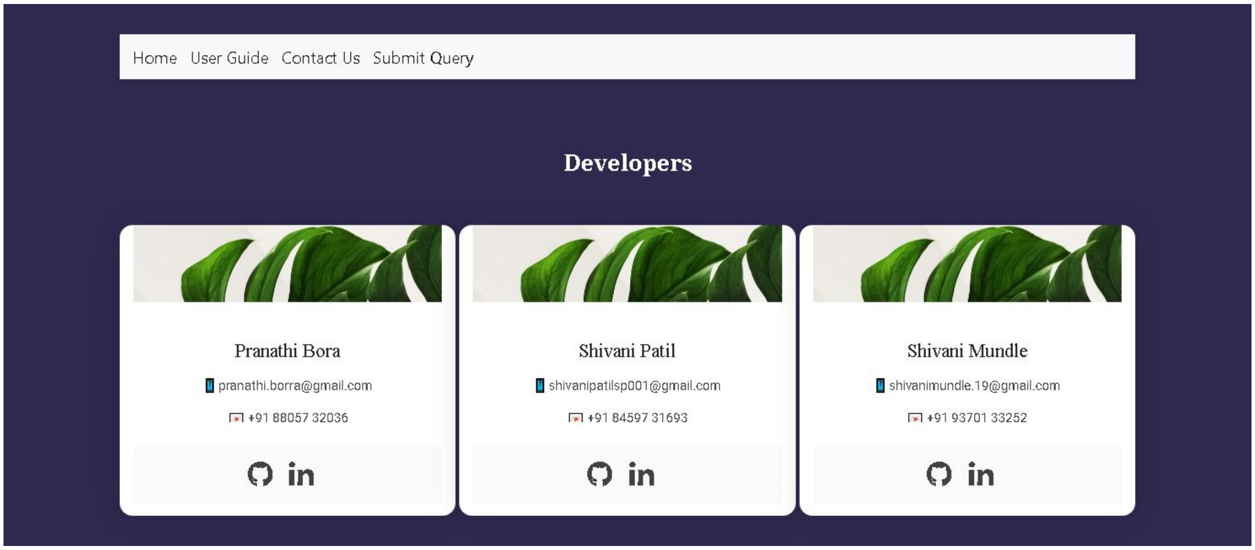
  6. **Hadoop NameNode UI**

     This will open the Hadoop NameNode UI where you can view information about the namenode and also see your directories and files on hdfs file system.

  7. **Submit Another Query**

     You will be redirected to the submit query page to run another query.

3. Contact Us Page



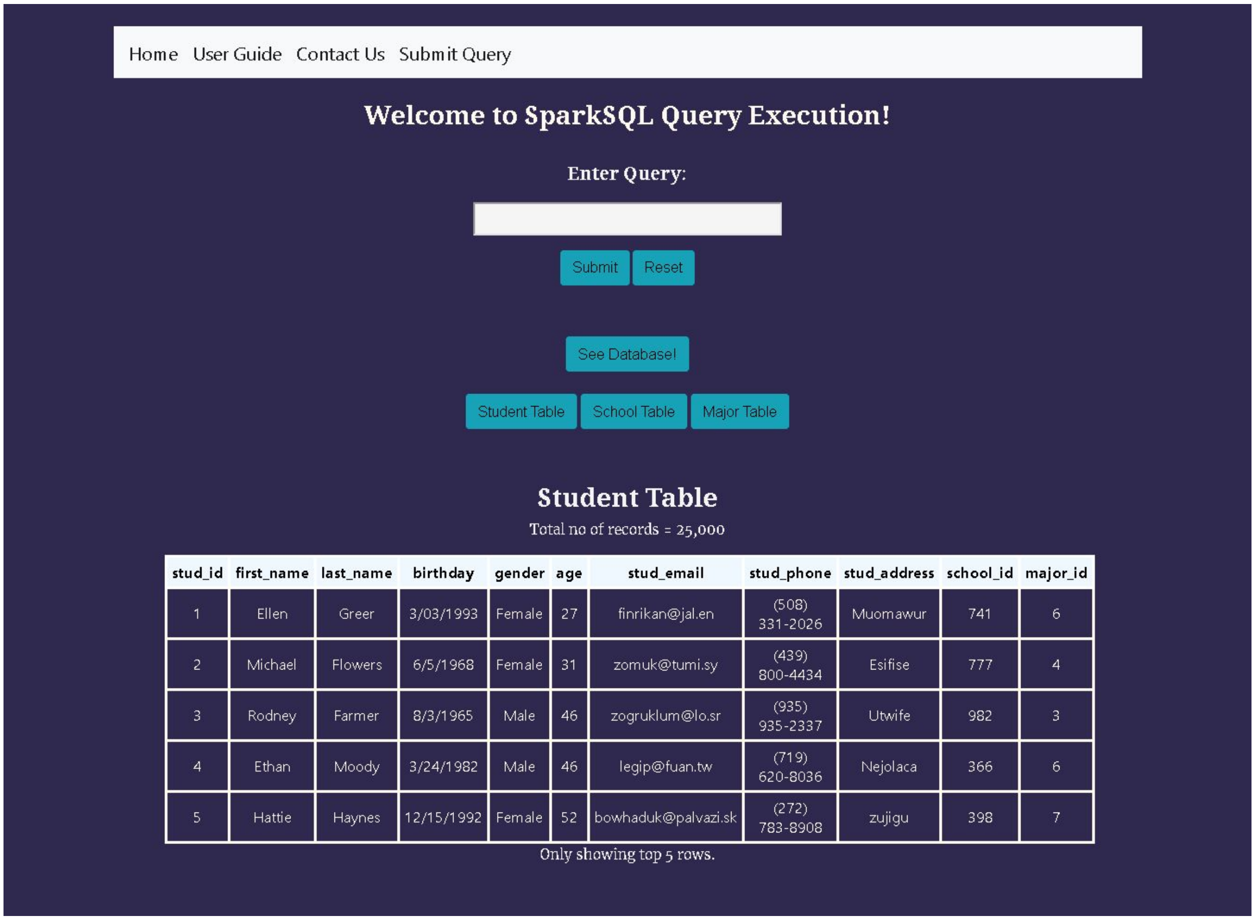
Developers

Pranathi Bora
pranathi.borra@gmail.com
+91 88057 32036

Shivani Patil
shivanipatilsp001@gmail.com
+91 84597 31693

Shivani Mundle
shivanimundle.19@gmail.com
+91 93701 33252

4. Analysis tool



Welcome to SparkSQL Query Execution!

Enter Query:

Student Table

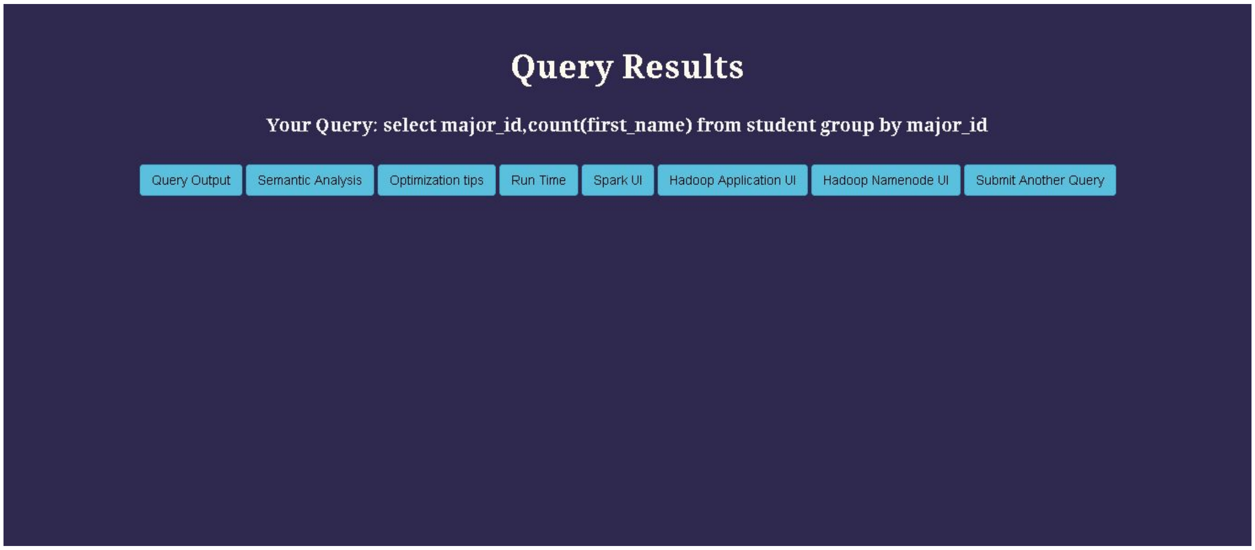
Total no of records = 25,000

stud_id	first_name	last_name	birthday	gender	age	stud_email	stud_phone	stud_address	school_id	major_id
1	Ellen	Greer	3/03/1993	Female	27	finrikan@jal.en	(508) 331-2026	Muomawur	741	6
2	Michael	Flowers	6/5/1968	Female	31	zomuk@tumi.sy	(439) 800-4434	Esifise	777	4
3	Rodney	Farmer	8/3/1965	Male	46	zogruklum@lo.sr	(935) 935-2337	Utwife	982	3
4	Ethan	Moody	3/24/1982	Male	46	legip@fuan.tw	(719) 620-8036	Nejolaca	366	6
5	Hattie	Haynes	12/15/1992	Female	52	bowhaduk@palvazi.sk	(272) 783-8908	zujigu	398	7

Only showing top 5 rows.

5. Analysis

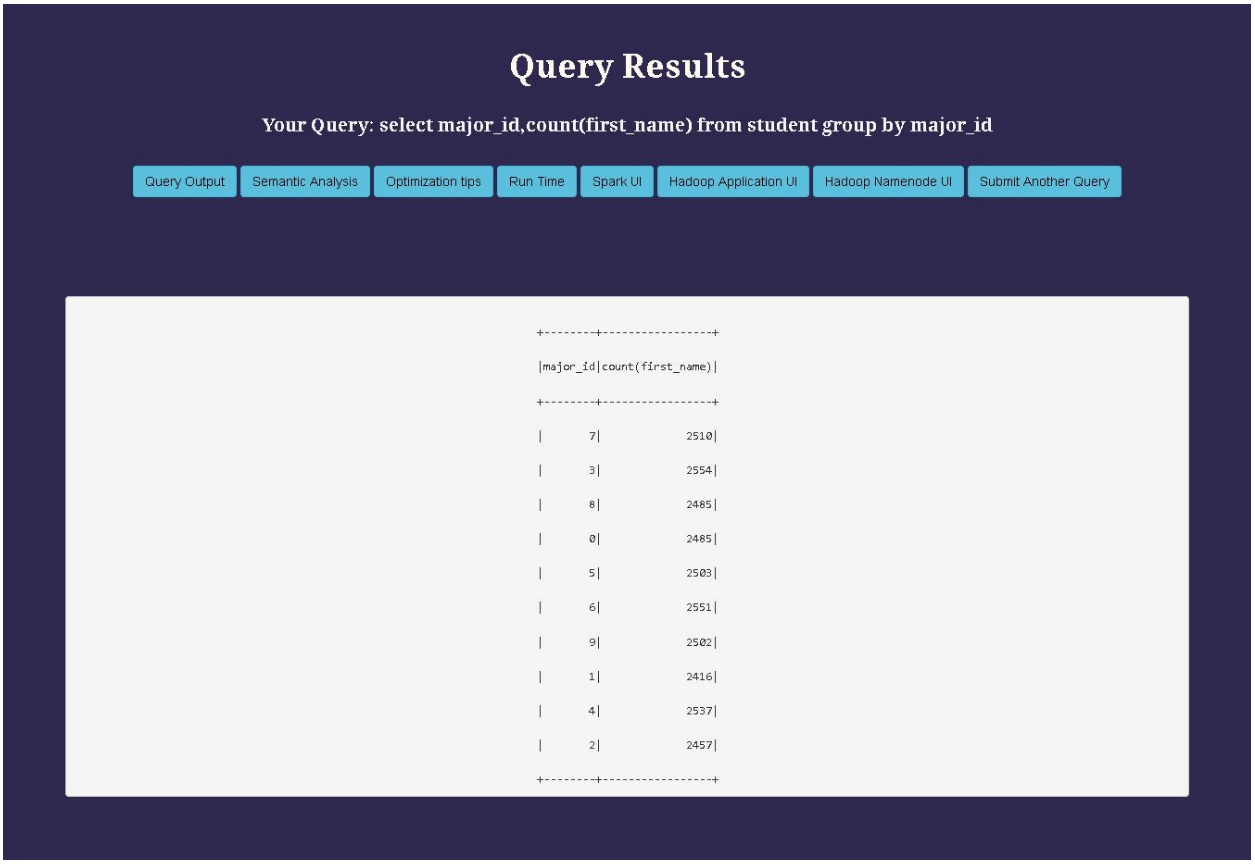
a. Submitted query



Query Results

Your Query: select major_id,count(first_name) from student group by major_id

b. Query Output



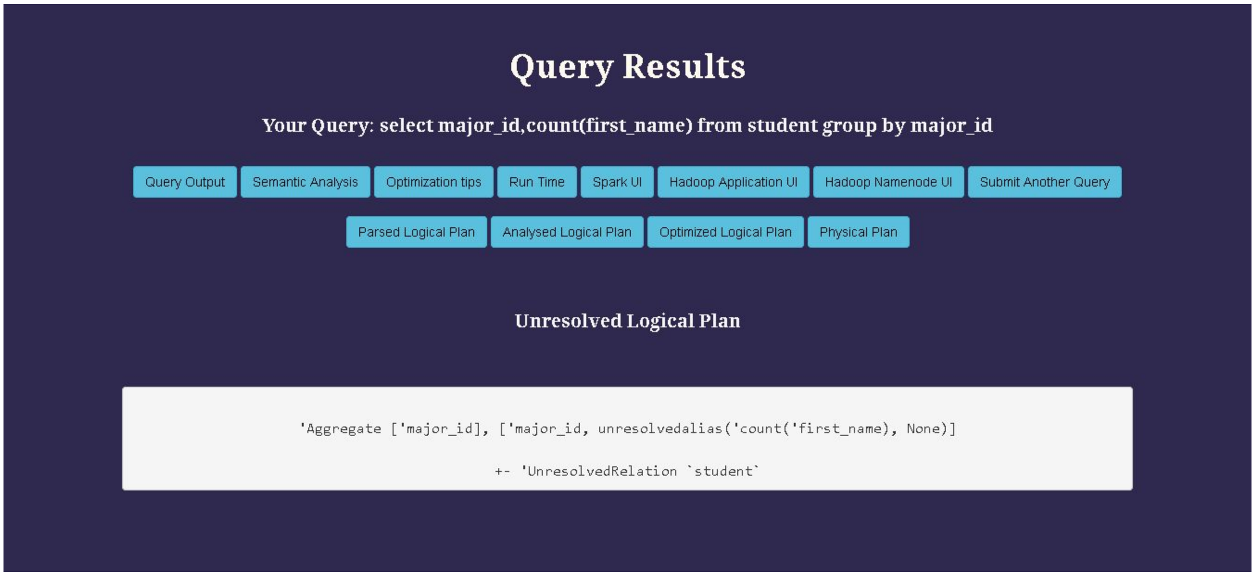
Query Results

Your Query: select major_id,count(first_name) from student group by major_id

```
+--------+----------------+
|major_id|count(first_name)|
+--------+----------------+
|       7|            2510|
|       3|            2554|
|       8|            2485|
|       0|            2485|
|       5|            2503|
|       6|            2551|
|       9|            2502|
|       1|            2416|
|       4|            2537|
|       2|            2457|
+--------+----------------+
```

c. Execution Plans

1. Unresolved Logical Plan



Query Results

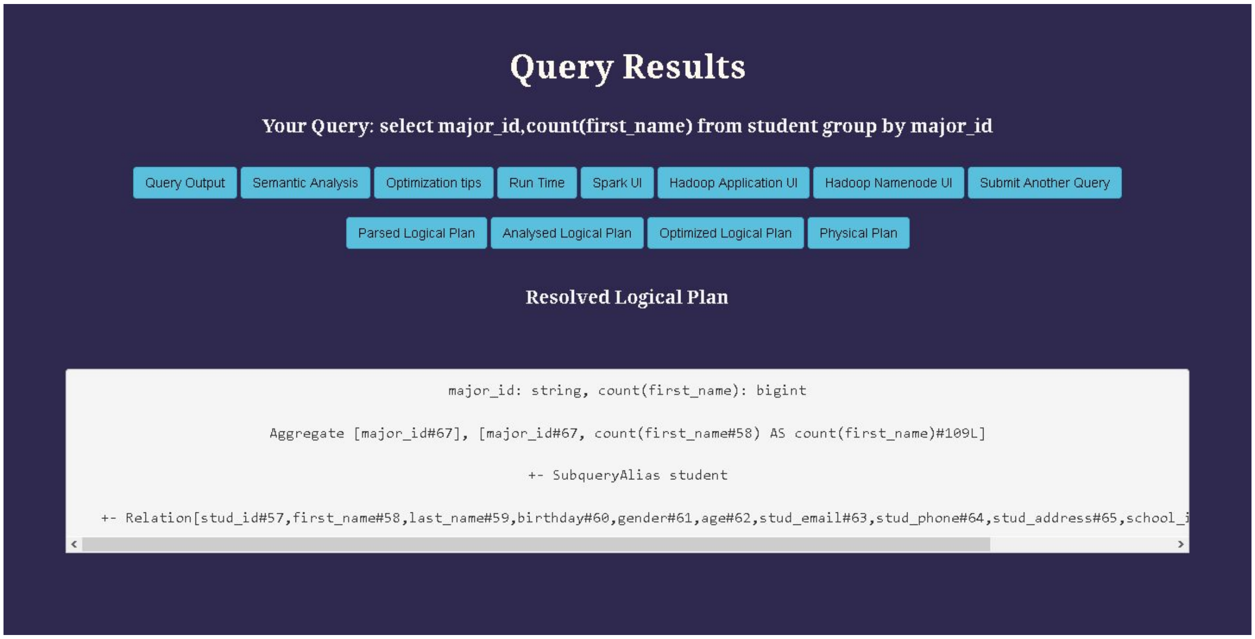
Your Query: select major_id,count(first_name) from student group by major_id

Unresolved Logical Plan

```
'Aggregate ['major_id], ['major_id, unresolvedalias('count('first_name), None)]

+- 'UnresolvedRelation `student`
```

2. Resolved Logical Plan



Query Results

Your Query: select major_id,count(first_name) from student group by major_id

Resolved Logical Plan

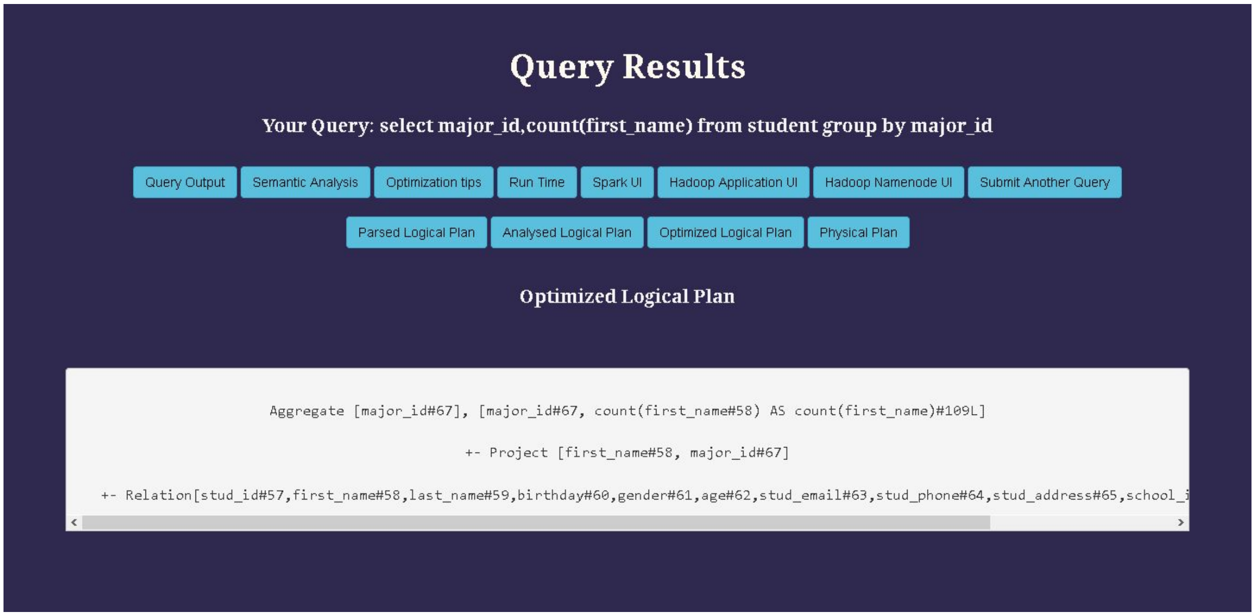
```
major_id: string, count(first_name): bigint

Aggregate [major_id#67], [major_id#67, count(first_name#58) AS count(first_name)#109L]

+- SubqueryAlias student

+- Relation[stud_id#57,first_name#58,last_name#59,birthday#60,gender#61,age#62,stud_email#63,stud_phone#64,stud_address#65,school_i
```

3. Optimized Logical Plan



Query Results

Your Query: select major_id,count(first_name) from student group by major_id

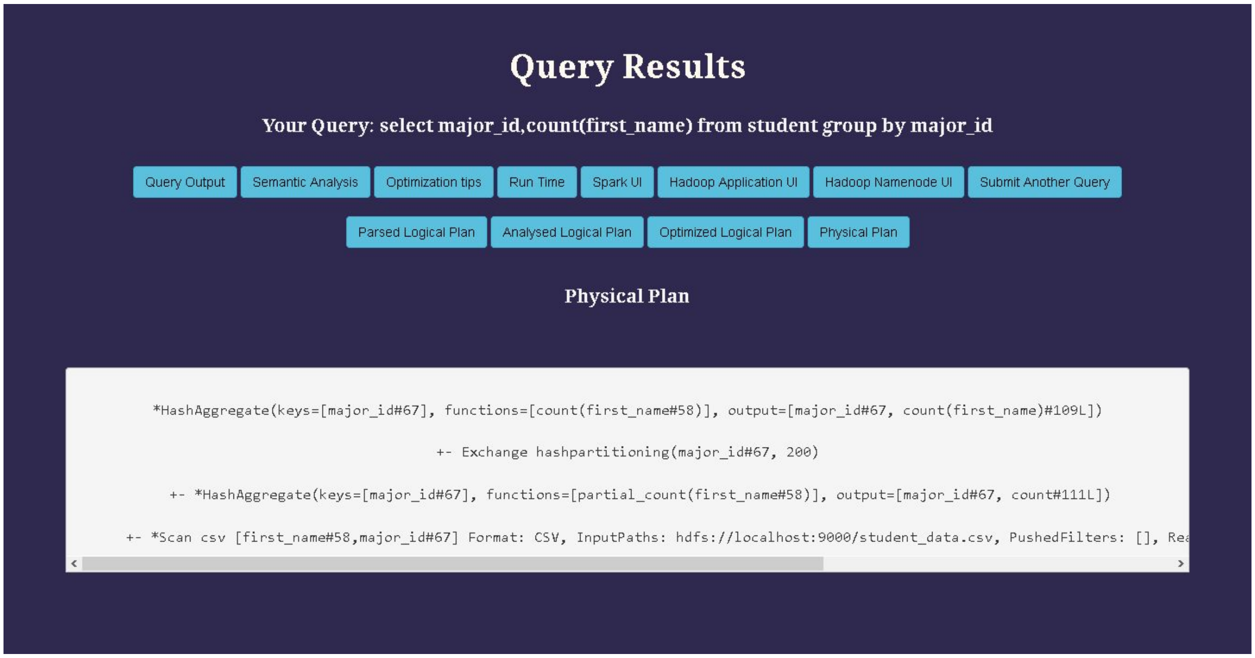
Optimized Logical Plan

```
Aggregate [major_id#67], [major_id#67, count(first_name#58) AS count(first_name)#109L]

+- Project [first_name#58, major_id#67]

+- Relation[stud_id#57,first_name#58,last_name#59,birthday#60,gender#61,age#62,stud_email#63,stud_phone#64,stud_address#65,school_i
```

4. Physical Plan



Query Results

Your Query: select major_id,count(first_name) from student group by major_id

Physical Plan

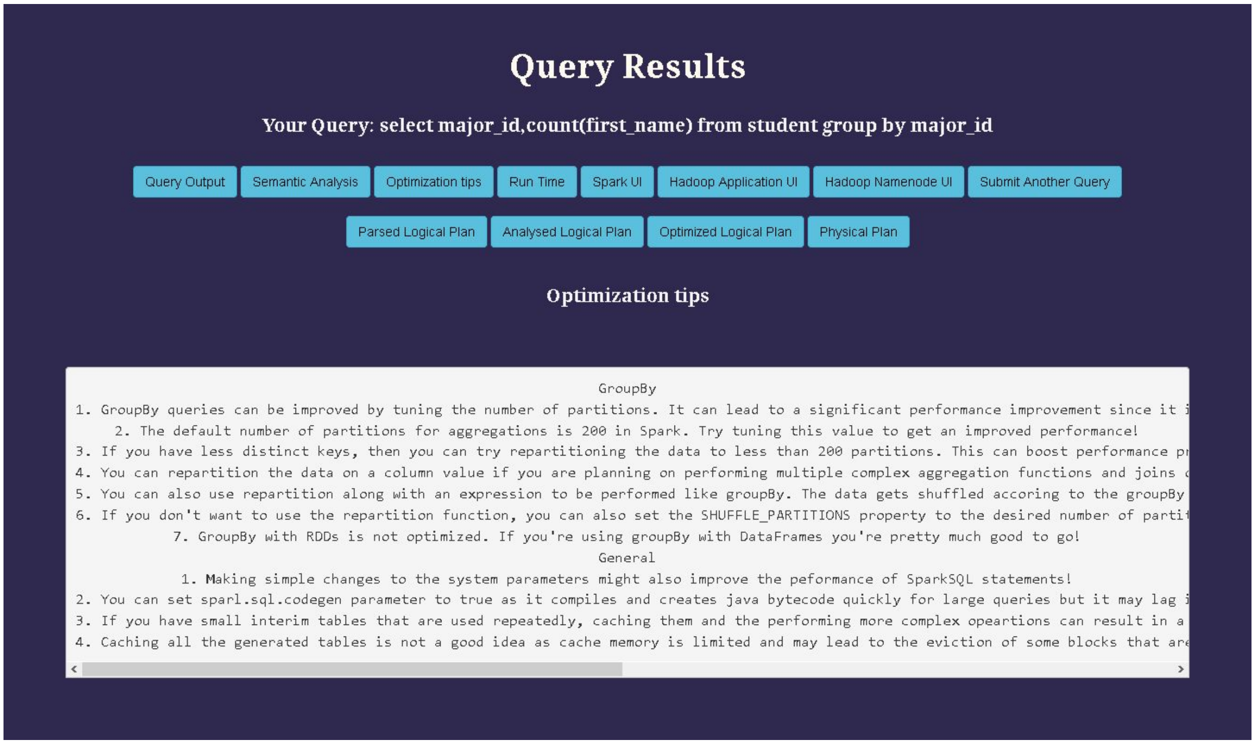
```
*HashAggregate(keys=[major_id#67], functions=[count(first_name#58)], output=[major_id#67, count(first_name)#109L])

+- Exchange hashpartitioning(major_id#67, 200)

+- *HashAggregate(keys=[major_id#67], functions=[partial_count(first_name#58)], output=[major_id#67, count#111L])

+- *Scan csv [first_name#58,major_id#67] Format: CSV, InputPaths: hdfs://localhost:9000/student_data.csv, PushedFilters: [], Rea
```

d. Optimization tips



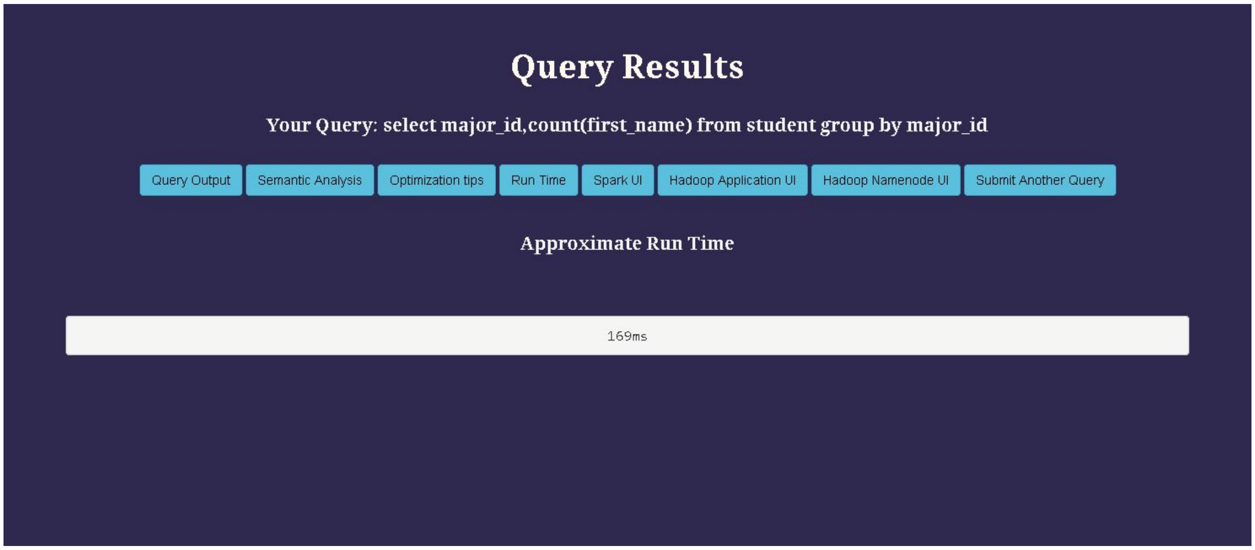
Query Results

Your Query: select major_id,count(first_name) from student group by major_id

Optimization tips

```
                                        GroupBy
1. GroupBy queries can be improved by tuning the number of partitions. It can lead to a significant performance improvement since it i
   2. The default number of partitions for aggregations is 200 in Spark. Try tuning this value to get an improved performance!
3. If you have less distinct keys, then you can try repartitioning the data to less than 200 partitions. This can boost performance pr
4. You can repartition the data on a column value if you are planning on performing multiple complex aggregation functions and joins
5. You can also use repartition along with an expression to be performed like groupBy. The data gets shuffled accoring to the groupBy
6. If you don't want to use the repartition function, you can also set the SHUFFLE_PARTITIONS property to the desired number of partit
          7. GroupBy with RDDs is not optimized. If you're using groupBy with DataFrames you're pretty much good to go!
                                        General
           1. Making simple changes to the system parameters might also improve the peformance of SparkSQL statements!
2. You can set sparl.sql.codegen parameter to true as it compiles and creates java bytecode quickly for large queries but it may lag i
3. If you have small interim tables that are used repeatedly, caching them and the performing more complex opeartions can result in a
4. Caching all the generated tables is not a good idea as cache memory is limited and may lead to the eviction of some blocks that are
```

e. Approximate run time



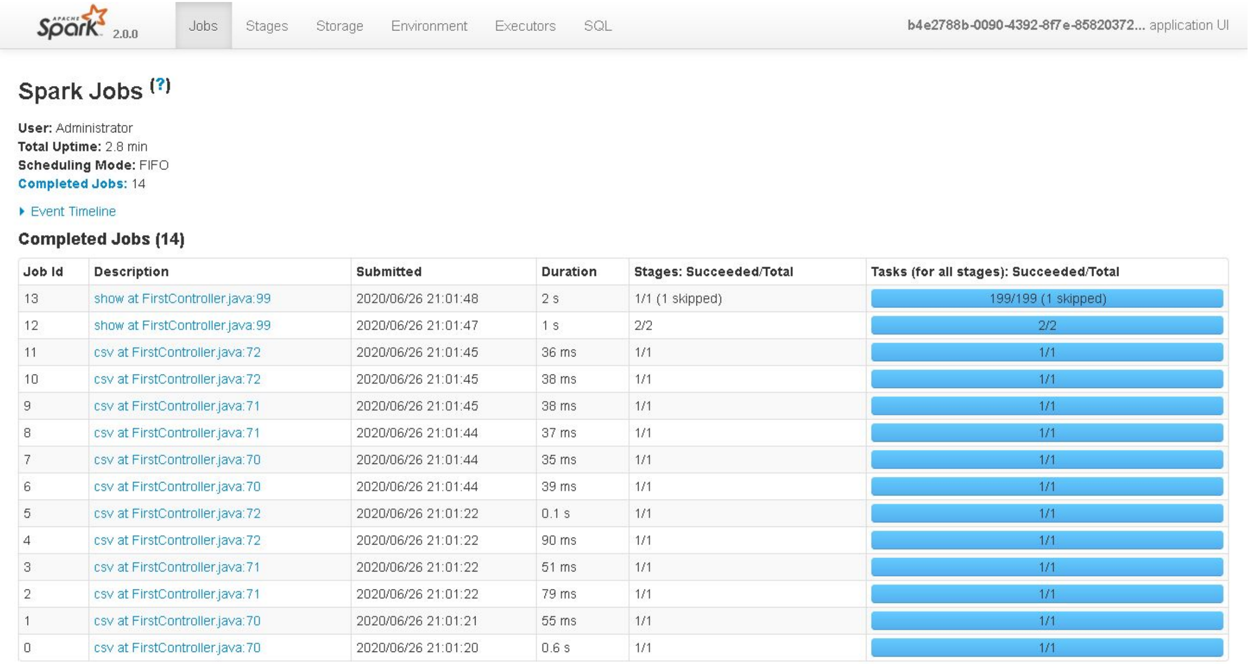
Query Results

Your Query: select major_id,count(first_name) from student group by major_id

Approximate Run Time

```
169ms
```

f. Spark UI



Spark Jobs (?)

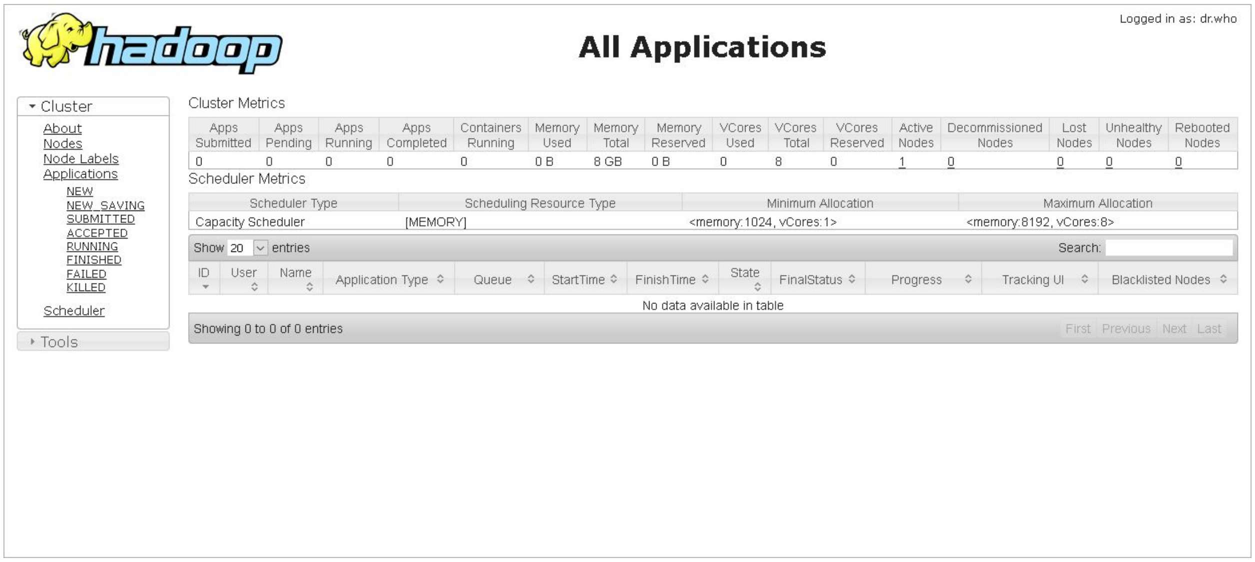
User: Administrator
Total Uptime: 2.8 min
Scheduling Mode: FIFO
Completed Jobs: 14

▸ Event Timeline

Completed Jobs (14)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
13	show at FirstController.java:99	2020/06/26 21:01:48	2 s	1/1 (1 skipped)	199/199 (1 skipped)
12	show at FirstController.java:99	2020/06/26 21:01:47	1 s	2/2	2/2
11	csv at FirstController.java:72	2020/06/26 21:01:45	36 ms	1/1	1/1
10	csv at FirstController.java:72	2020/06/26 21:01:45	38 ms	1/1	1/1
9	csv at FirstController.java:71	2020/06/26 21:01:45	38 ms	1/1	1/1
8	csv at FirstController.java:71	2020/06/26 21:01:44	37 ms	1/1	1/1
7	csv at FirstController.java:70	2020/06/26 21:01:44	35 ms	1/1	1/1
6	csv at FirstController.java:70	2020/06/26 21:01:44	39 ms	1/1	1/1
5	csv at FirstController.java:72	2020/06/26 21:01:22	0.1 s	1/1	1/1
4	csv at FirstController.java:72	2020/06/26 21:01:22	90 ms	1/1	1/1
3	csv at FirstController.java:71	2020/06/26 21:01:22	51 ms	1/1	1/1
2	csv at FirstController.java:71	2020/06/26 21:01:22	79 ms	1/1	1/1
1	csv at FirstController.java:70	2020/06/26 21:01:21	55 ms	1/1	1/1
0	csv at FirstController.java:70	2020/06/26 21:01:20	0.6 s	1/1	1/1

g. Hadoop application UI



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0	0	0	0	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0

Scheduler Metrics

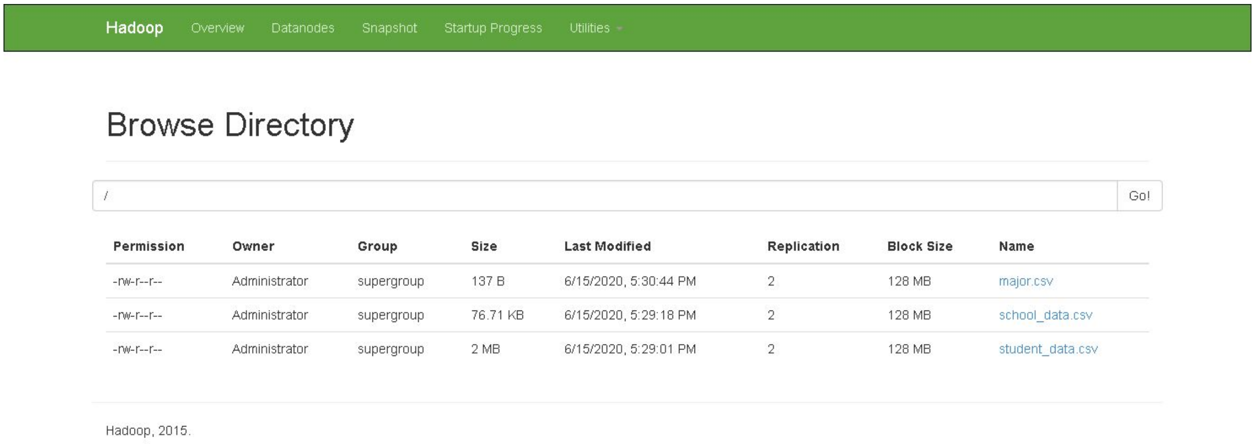
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
No data available in table											

Showing 0 to 0 of 0 entries First Previous Next Last

h. Namenode UI



Browse Directory

/

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	Administrator	supergroup	137 B	6/15/2020, 5:30:44 PM	2	128 MB	major.csv
-rw-r--r--	Administrator	supergroup	76.71 KB	6/15/2020, 5:29:18 PM	2	128 MB	school_data.csv
-rw-r--r--	Administrator	supergroup	2 MB	6/15/2020, 5:29:01 PM	2	128 MB	student_data.csv

Hadoop, 2015.