Big Data Programming

## CSEE5590/490

**Lab 1 Report**

Submitted By:

Sneha Mishra:

Class ID: 11

Aditya Soman

Class ID: 19

## Links:

**Github:**

1. [**https://github.com/adityasomann/Big-Data-Programming/wiki/Lab-1**](https://github.com/adityasomann/Big-Data-Programming/wiki/Lab-1)

**2.<https://github.com/SnehaMishra28/BigData_Programming_Summer2018/wiki/Lab-%231>**

**Youtube:**

<https://youtu.be/5ih-MBgo7SY>

**Question 1:**

**Finding Facebook common friends**

Explanation:

Consider the following users: 1,2,3,4,5.

They have friends like this:

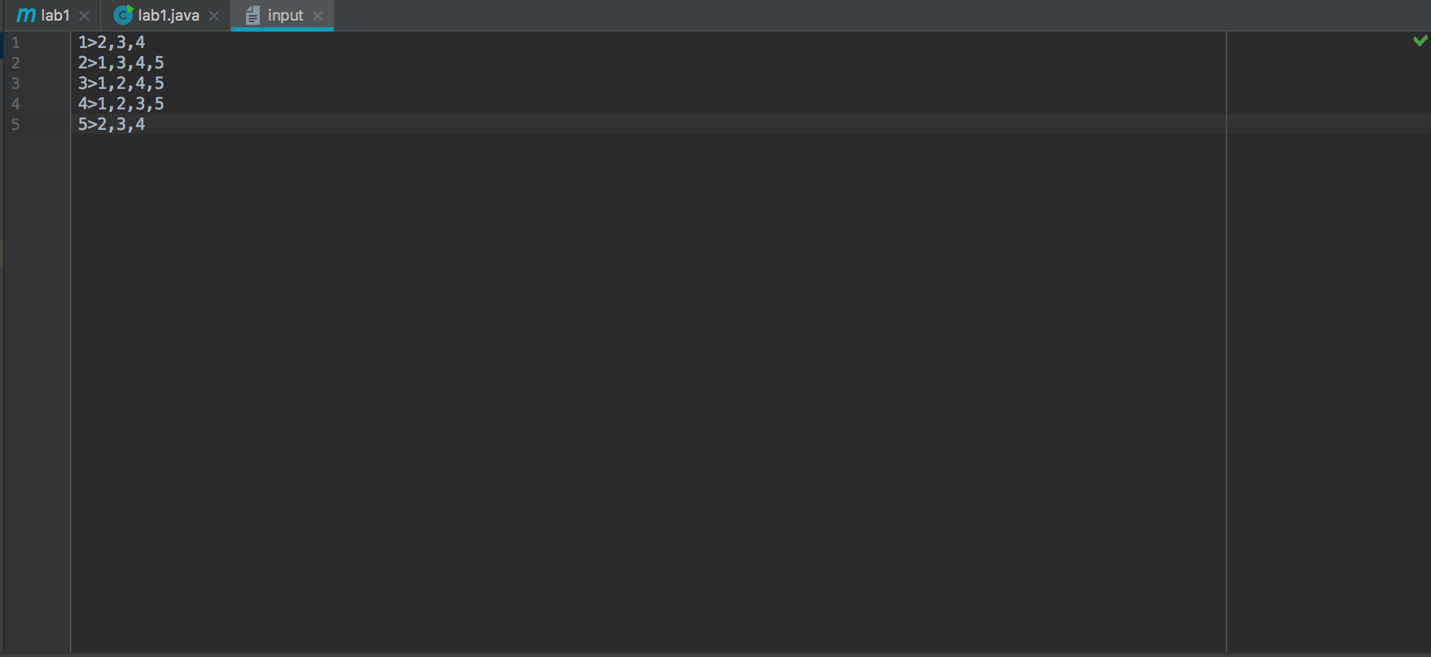
1>2,3,4 2>1,3,4,5 3>1,2,4,5 4>1,2,3,5 5>2,3,4

Using MapReduce, we have to figure out the mutual friends so that the output will be like:

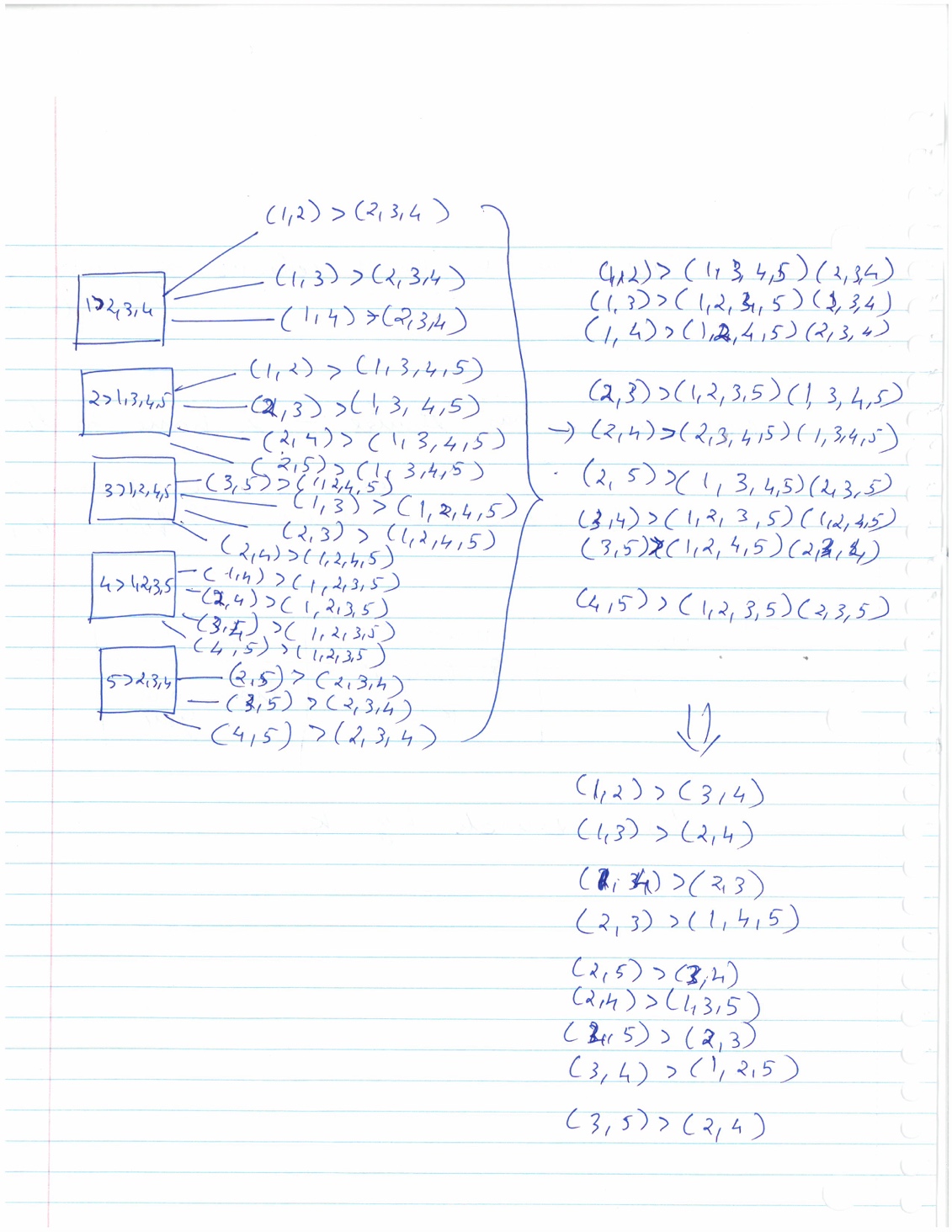
(1,2) (3,4) (1,3) (2,4) (1,4) (2,3) (2,3) (1,4,5) (2,4) (1,3,5) (2,5) (3,4) (3,4) (1,2,5) (3,5) (2,4) (4,5) (2,3)

**Screenshots:**

**Input File**

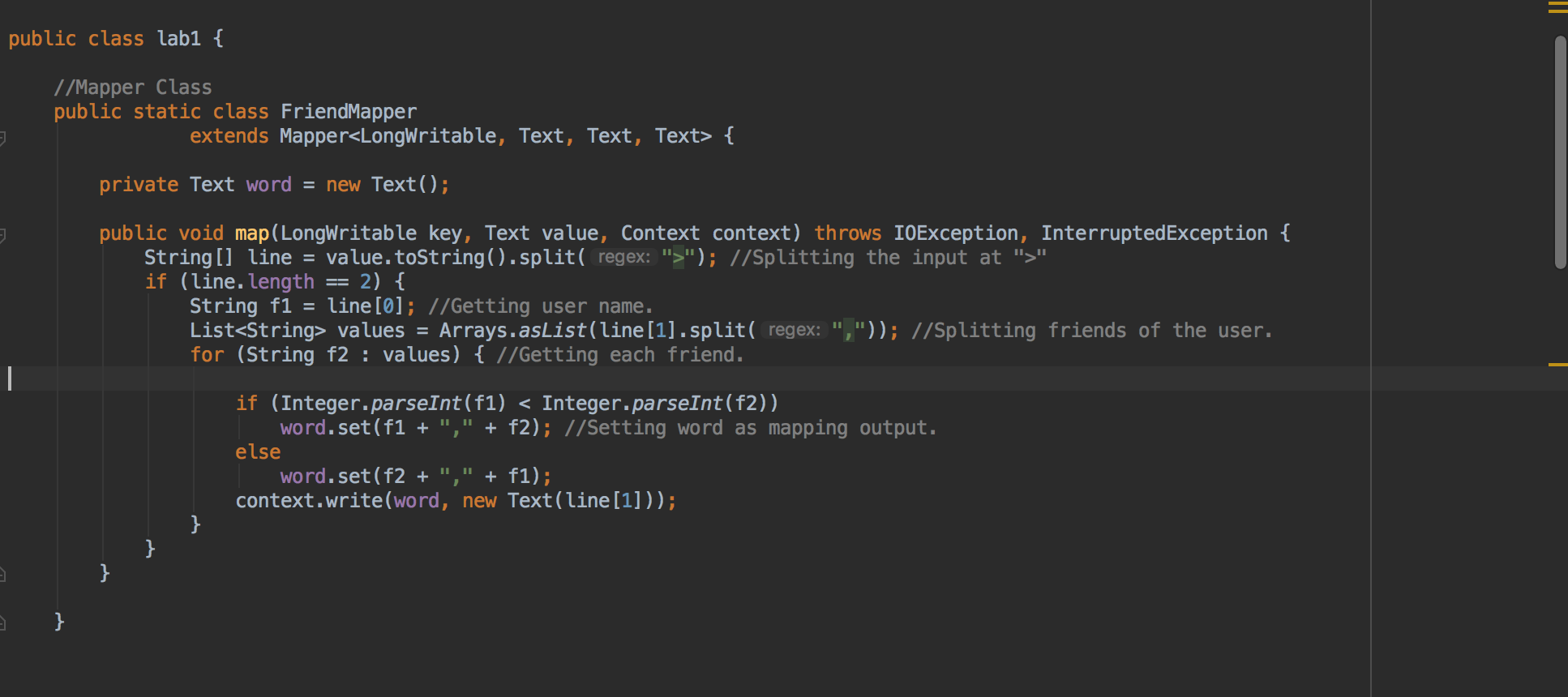


**MapReduce Diagram:**

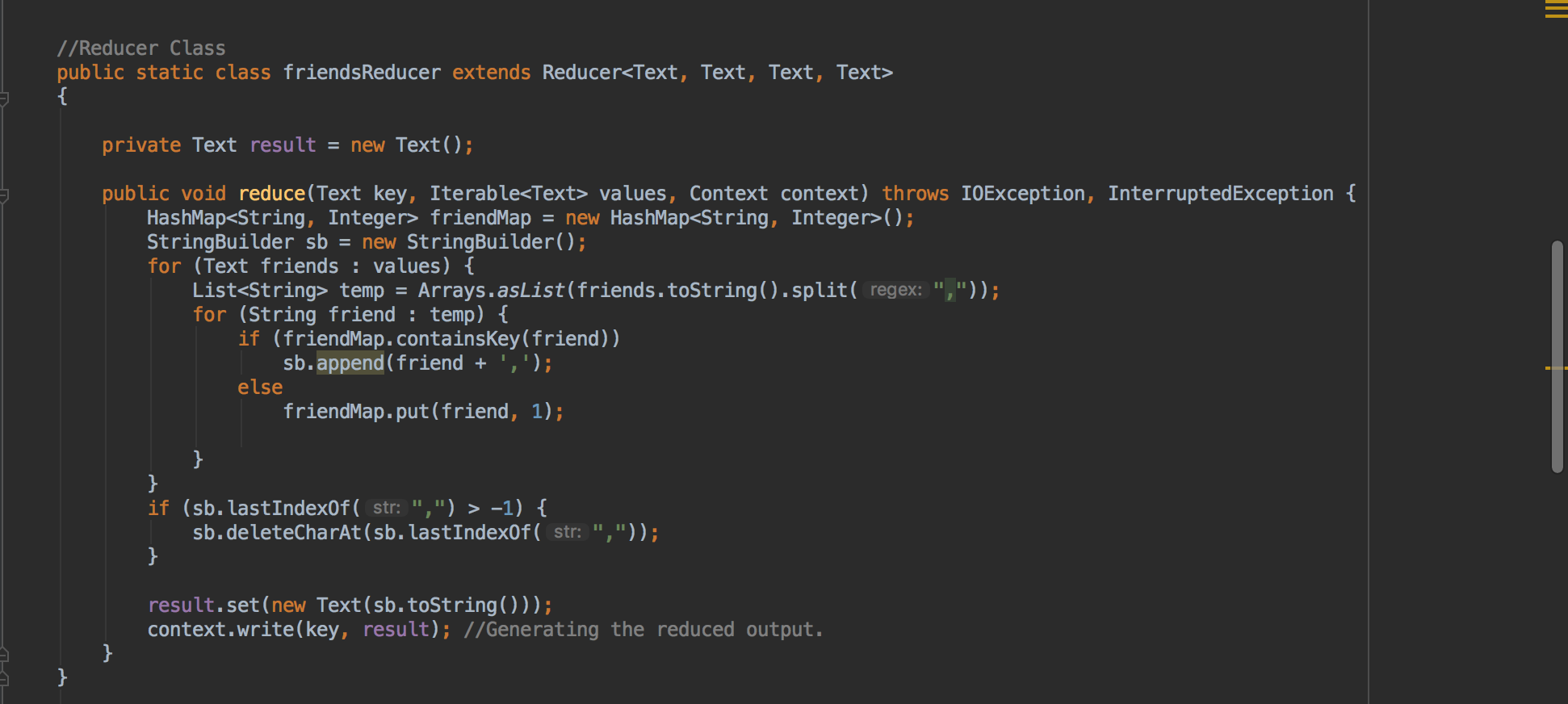


**Code Files:**

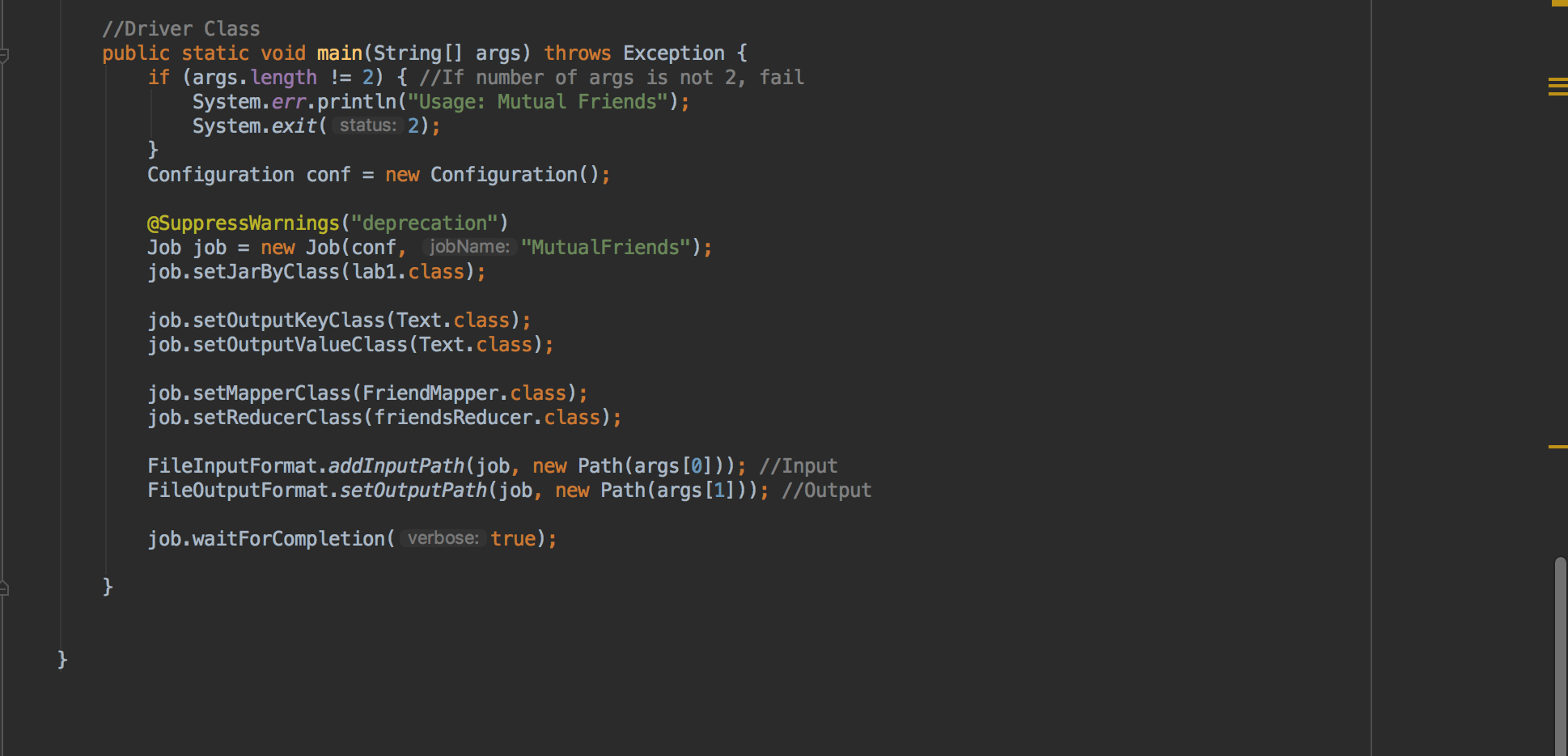
**Mapper Class:**



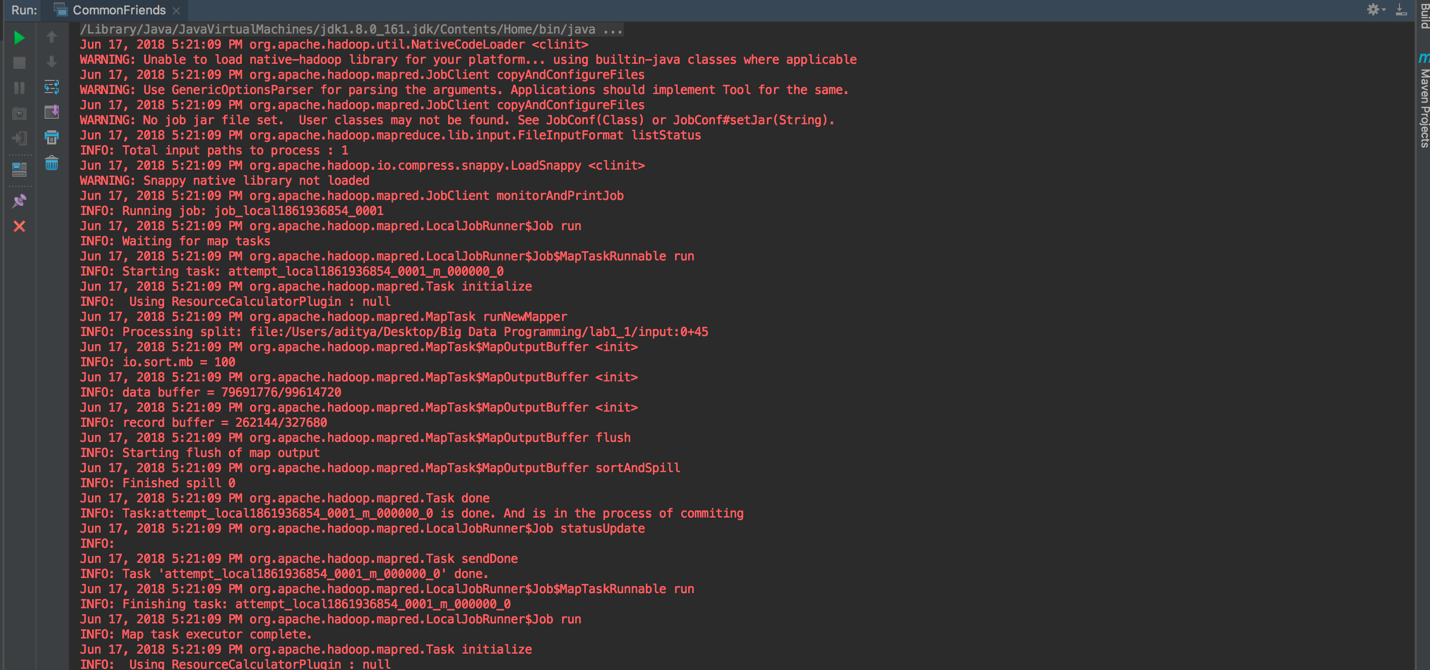
**Reducer Class:**



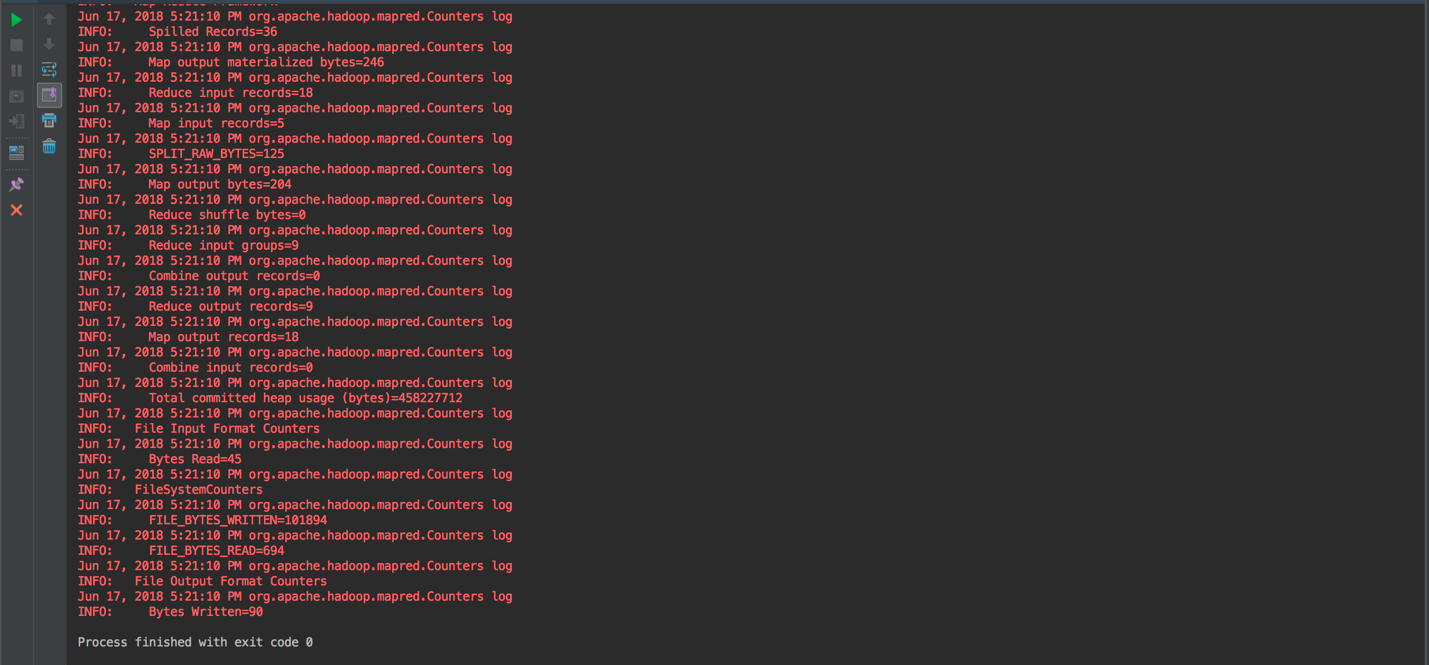
**Driver Class:**



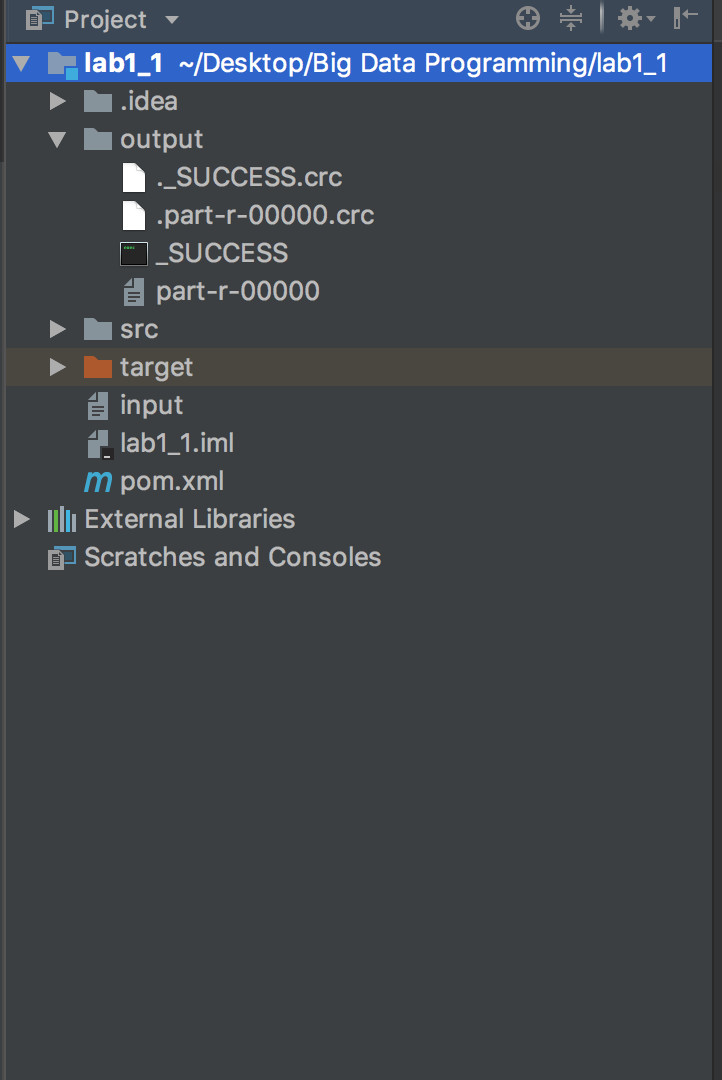
**Execution:**



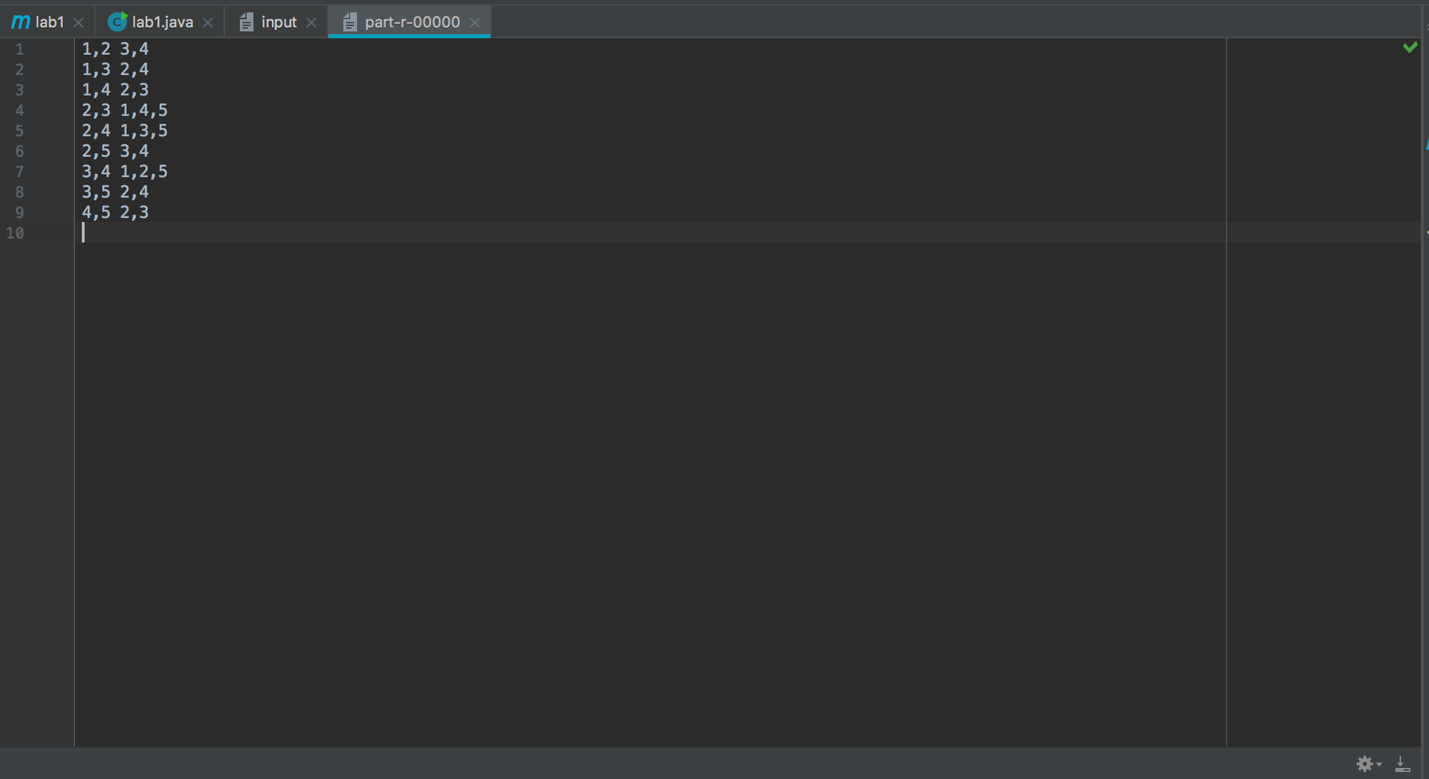
**Exit Code 0:**



**Output Generated:**

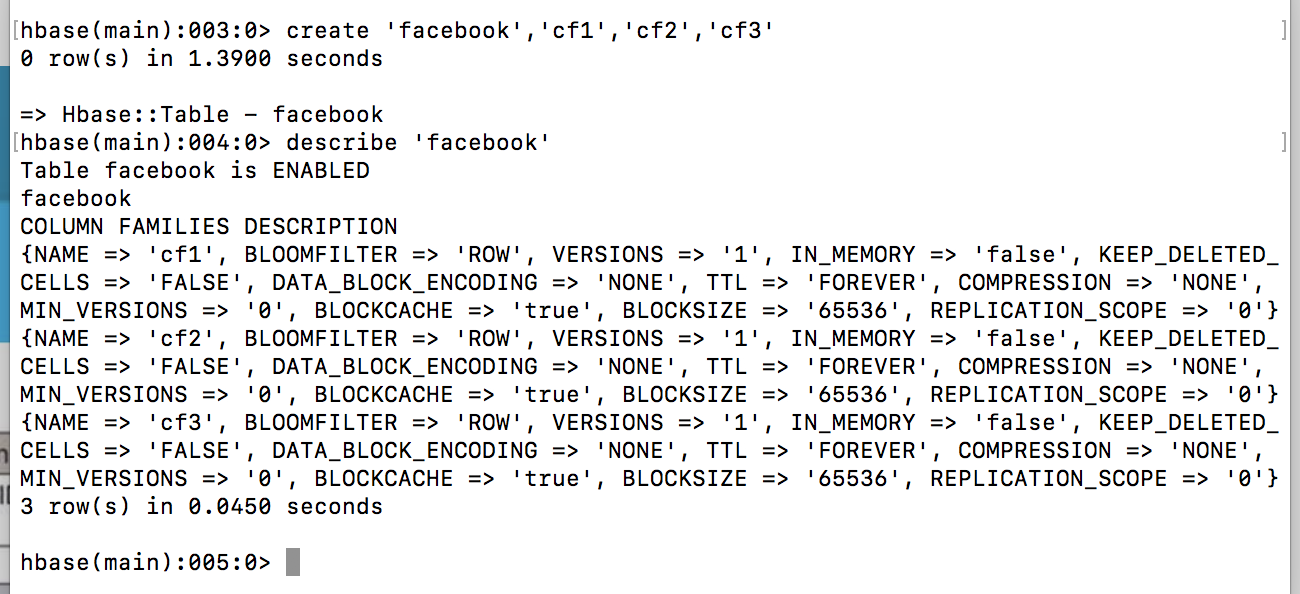


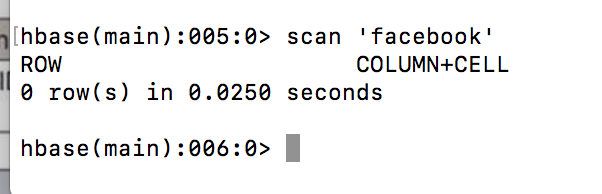
**Output File:**



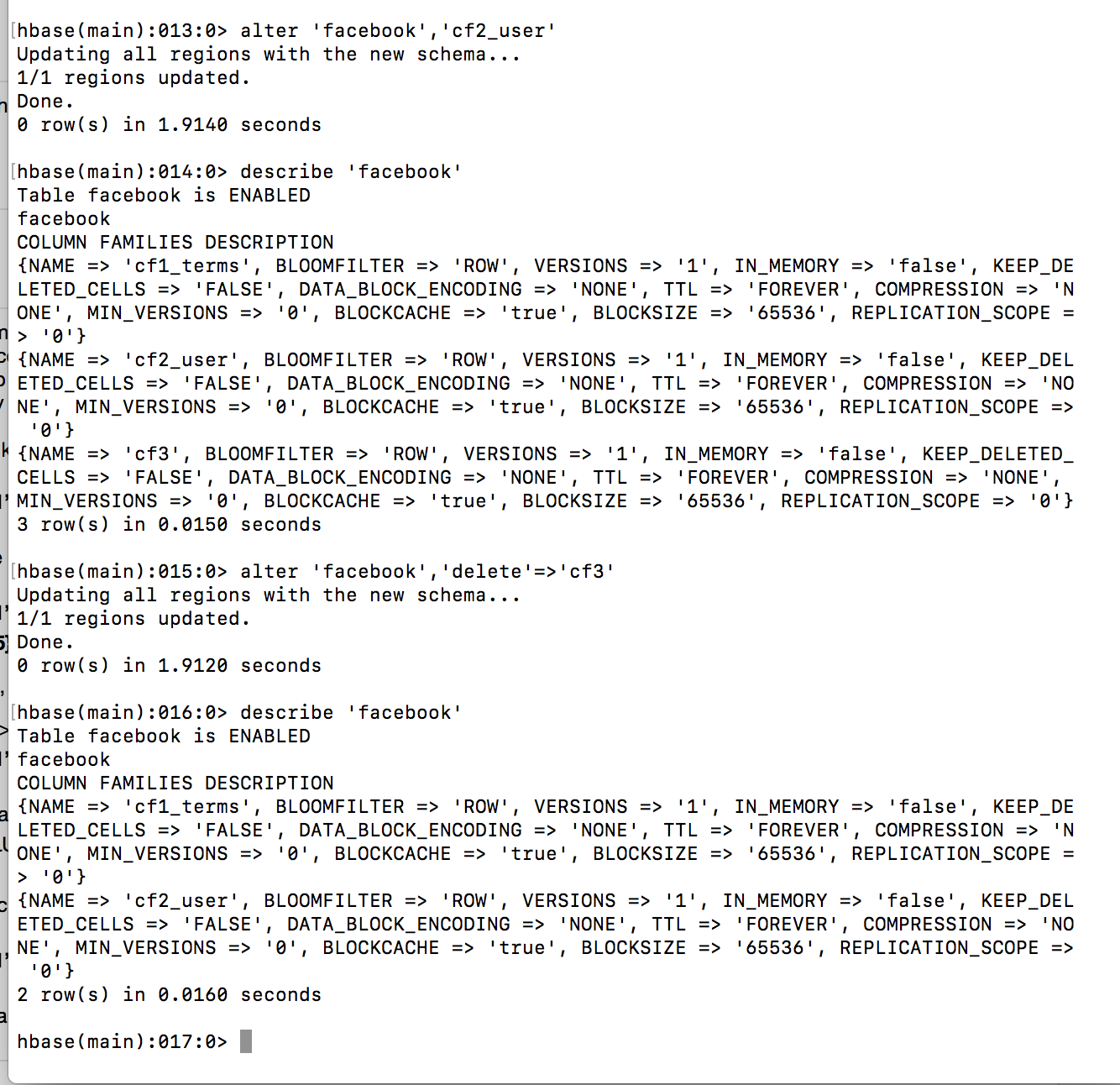
**Question 2**

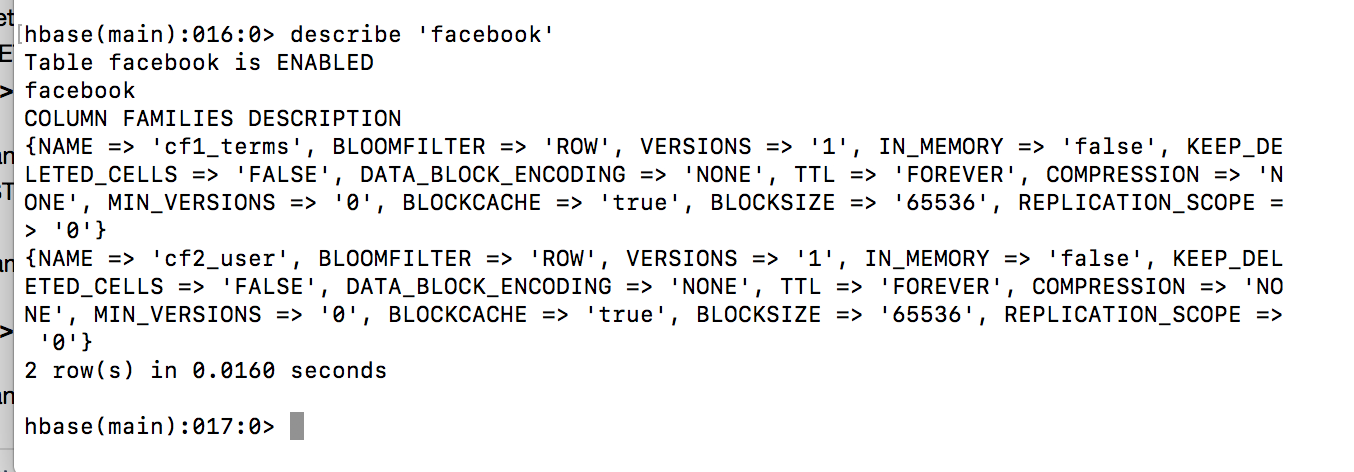
**Use Case: Facebook Messaging - Inbox/ Term Search**

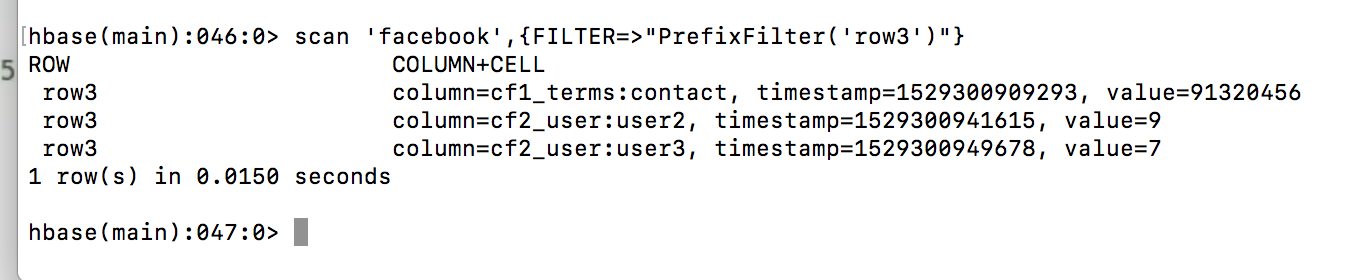
1. When you login to Facebook and click Messages => "See all messages", you will see a search box... you can search your inbox in there..  
   The basic idea is that you use the user id as the partition key, and then all the information you need for an inbox search will be clustered as rows in that partition. You can then set up multiple tables like this with different types of data clustered in the partition to support different types of searches. Since Cassandra can access a partition in essentially constant time even with millions of users, the system can scale and remain fast as you add nodes and users.
2. HBase implementation:  
   Step 1:  
   Create table named 'facebook'  
   

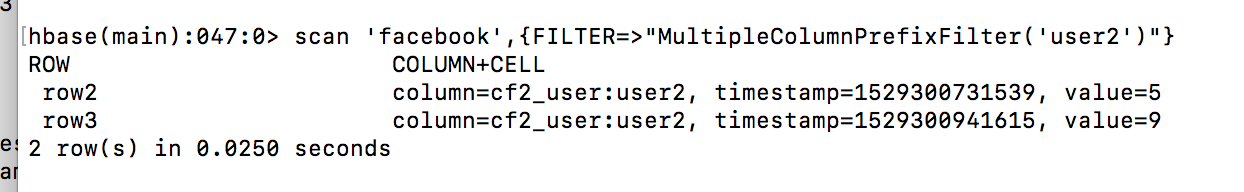
Empty Table 

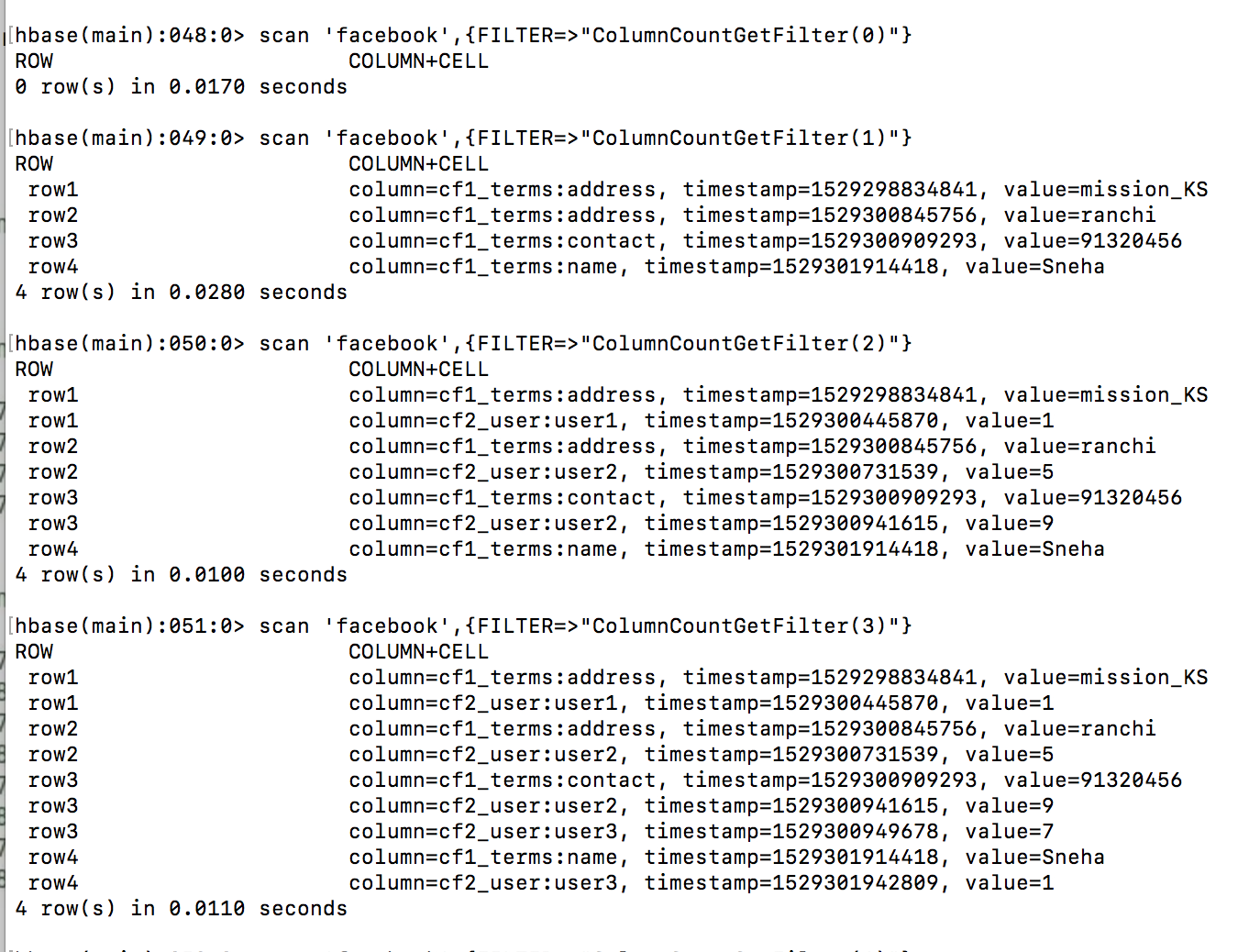
Update table (put values and alter columns) 

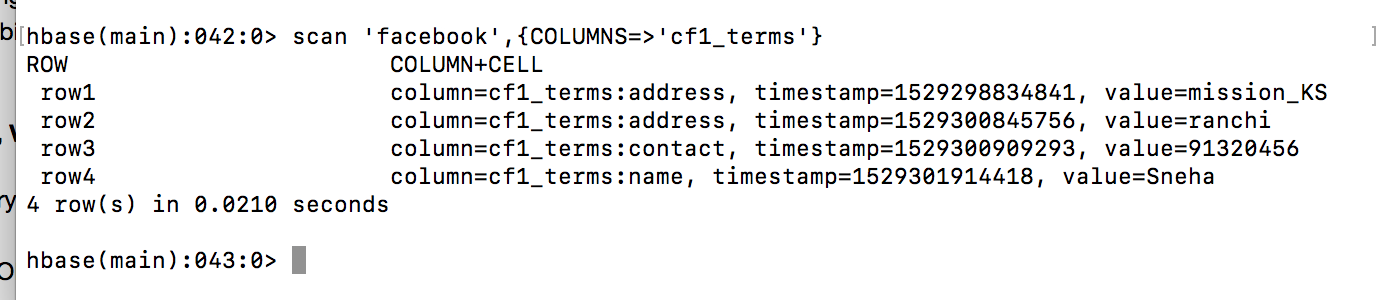


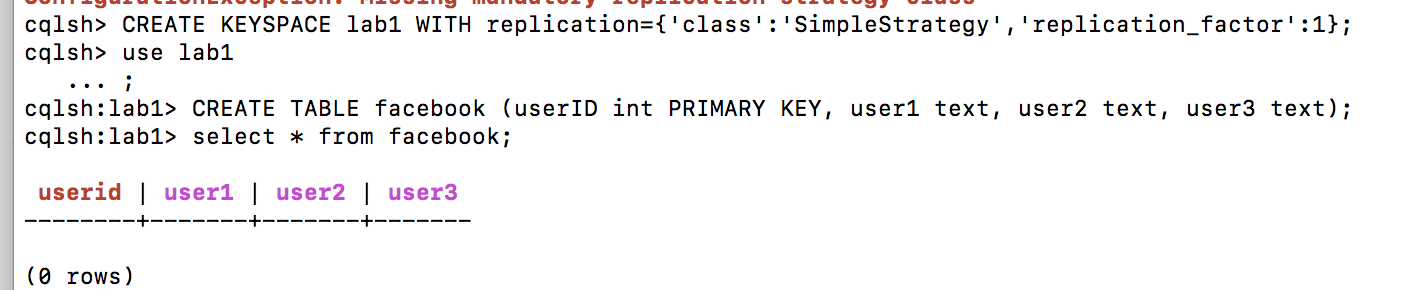
Describe Table details 

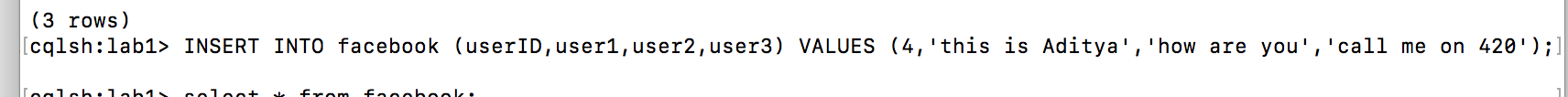
Step 2: Queries  
Query 1: PrefixFilter: This filter takes one argument as a prefix of a row key. It returns solely those key-values present in the very row that starts with the specified row prefix 

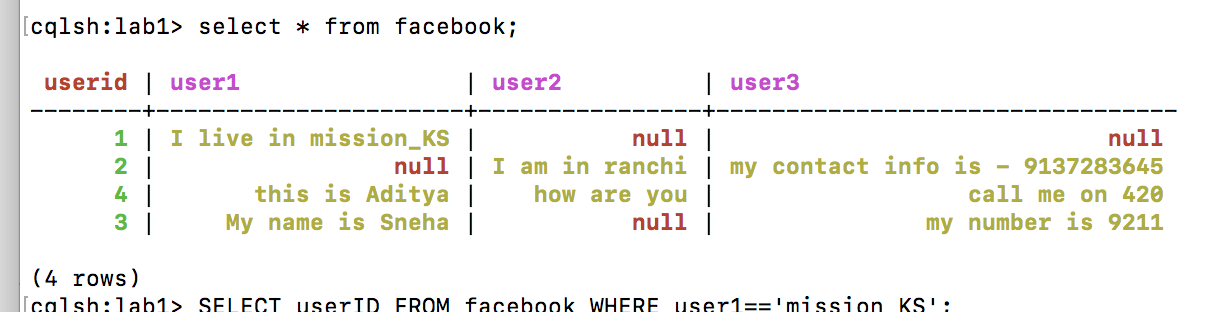
Query 2: MultipleColumnPrefixFilter: This filter takes a listing of column prefixes. It returns key-values that are present in the very column that starts with any of the specified column prefixes. every column prefixes should be a form qualifier. 

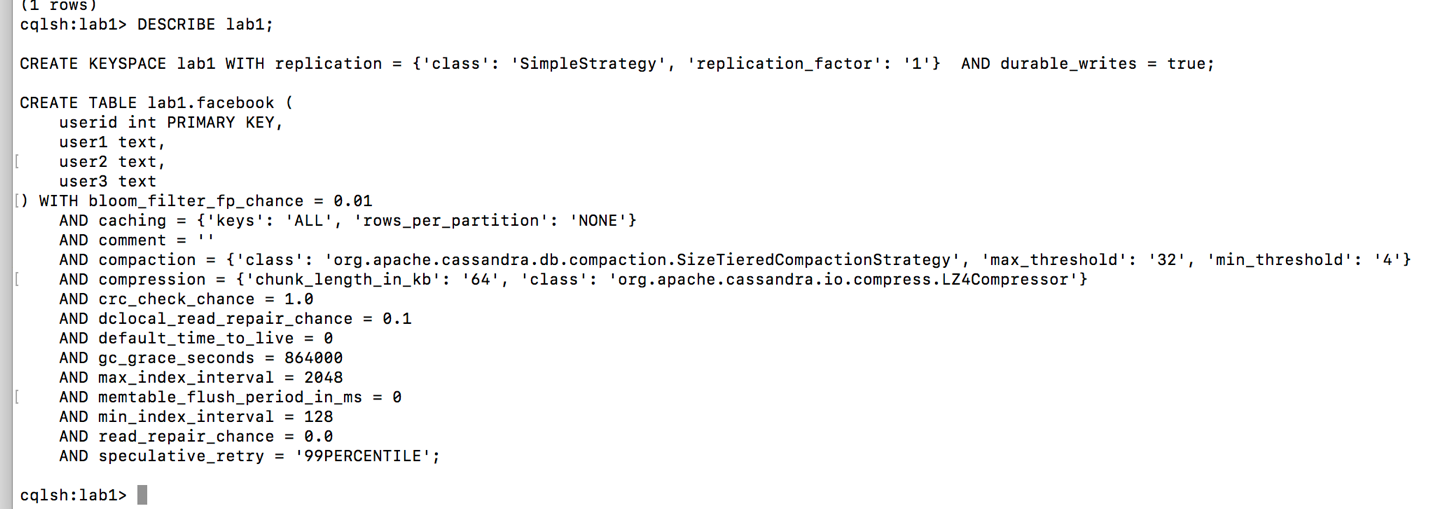
Query 3: ColumnCountGetFilter: This filter takes one argument a limit. It returns the primary limit number of columns within the table. 

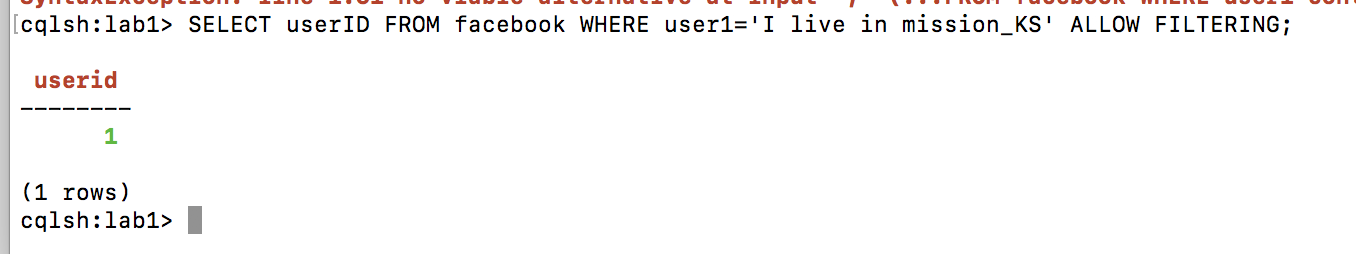
Query 4: Filter applied column family wise 

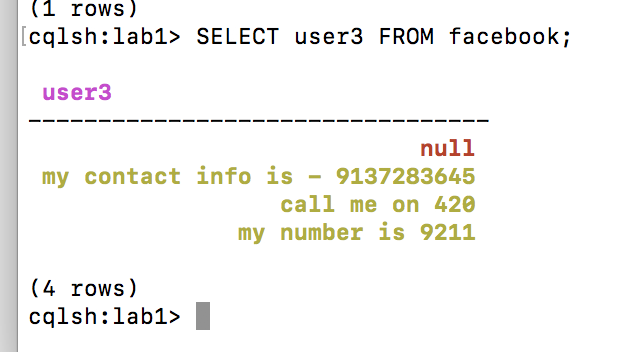
1. Cassandra implementation:  
   Step 1: Create Keyspace and Table named facebook The table facebook is created with messages from different Users as columns (user1, user2 etc). 

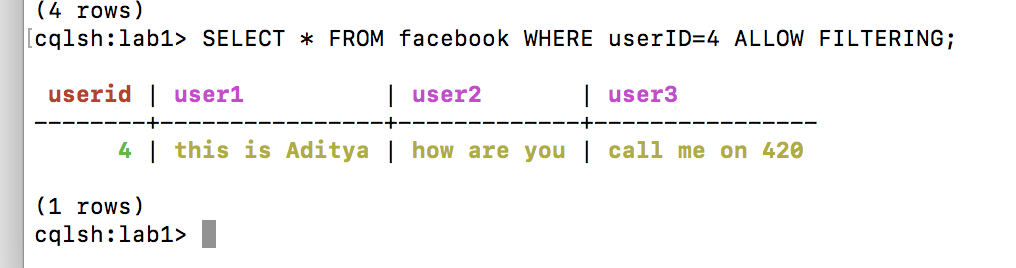
Insert data in the created Table 

Final updated Table with data 

Final created Keyspace description 

Step 2: Perform queries on the created facebook Table to get the desired output.  
Query 1: Term search - performs the text search to find the particular text sent by any user 

Query 2: User search - returns all the contents from the selected User 

Query 3: Interaction Search - returns all the conversation with all the users for a particular User ID 

1. Comparison of Cassandra and HBase based on the selected use case:

### Differences between Cassandra and HBase:

| **Name** | **Cassandra** | **HBase** |
| --- | --- | --- |
| Description | Wide-column store based on ideas of BigTable and DynamoDB | Wide-column store based on ideas of BigTable and DynamoDB |
| Primary database model | Wide column store | Wide column store |
| License | Open Source | Open Source |
| Cloud-based | No | NO |
| Implementation language | Java | Java |
| Server operating systems | BSD, Linux, OS X, Windows | Linux, Unix, Windows |
| Data scheme | Schema Free | Schema Free |
| Typing | Yes | No |
| XML support | No | No |
| Secondary indexes | Restricted | No |
| SQL | SQL-like DML and DDL statements (CQL) | No |
| APIs and other access methods | Proprietary protocol, Thrift | Java API, Restful HTTP API, Thrift |
| Supported programming languages | C#, C++, Clojure, Erlang, Go, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby, Scala | C, C#, C++, Groovy, Java, PHP, Python, Scala |
| Server-side scripts | No | Yes |
| Triggers | Yes | Yes |
| Partitioning methods | Sharding | Sharding |
| Replication methods | selectable replication factor | selectable replication factor |
| MapReduce | Yes | Yes |
| Consistency Concepts | Eventual Consistency, Immediate Consistency | Immediate Consistency |
| Foreign keys | No | No |
| Transaction concepts | No | No |
| Concurrency | Yes | Yes |
| Durability | Yes | Yes |
| In-memory capabilities | No | No |
| User concepts | Access rights for users can be defined per object | Access Control Lists (ACL) |

**References**

1. <https://stackoverflow.com/questions/28130774/how-did-facebook-use-cassandra-for-inbox-search-if-caasandra-has-no-search-capa>
2. <https://www.quora.com/What-is-inbox-search-on-Facebook>
3. <http://highscalability.com/blog/2010/11/16/facebooks-new-real-time-messaging-system-hbase-to-store-135.html>
4. <http://horicky.blogspot.com/2010/10/bigtable-model-with-cassandra-and-hbase.html>
5. <https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/>
6. <https://drill.apache.org/docs/querying-hbase/>
7. <https://acadgild.com/blog/different-types-of-filters-in-hbase-shell>
8. <https://db-engines.com/en>
9. <http://stevekrenzel.com/finding-friends-with-mapreduce>