Apache Spark Demo

연세대학교 컴퓨터과학과 박상현
2018년 10월

SW STOR LAB
Software Technology Advanced Research

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발
과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

IITP 정보통신기술진흥센터
Institute for Information & communications Technology Promotion

Contents

- Install Apache Spark
- Run spark-shell (Scala Interactive mode)
  - SparkContext
  - Transformation & Action
  - Example) Word count

Configuration

- Java v1.8

```
hwan@beluga:~$ java -version
java version "1.8.0_171"
Java(TM) SE Runtime Environment (build 1.8.0_171-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.171-b11, mixed mode)
```

- Scala v2.11

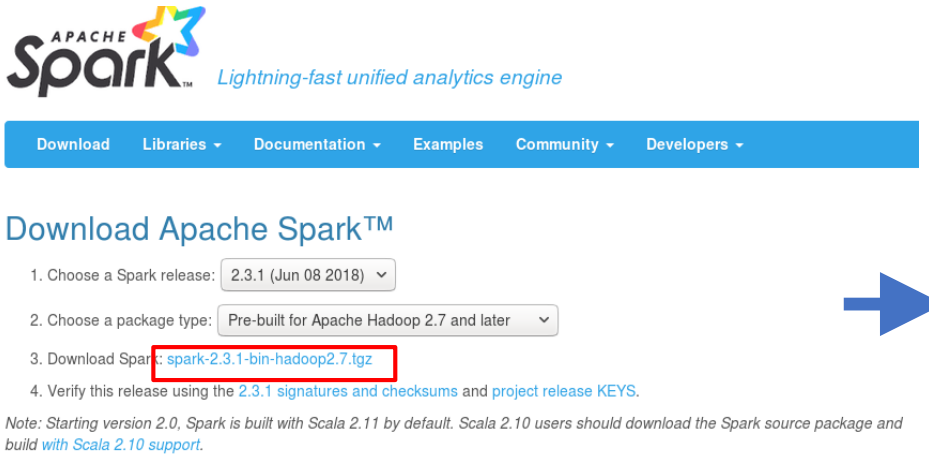
```
hwan@beluga:~$ scala -version
Scala code runner version 2.11.8 -- Copyright 2002-2016, LAMP/EPFL
```

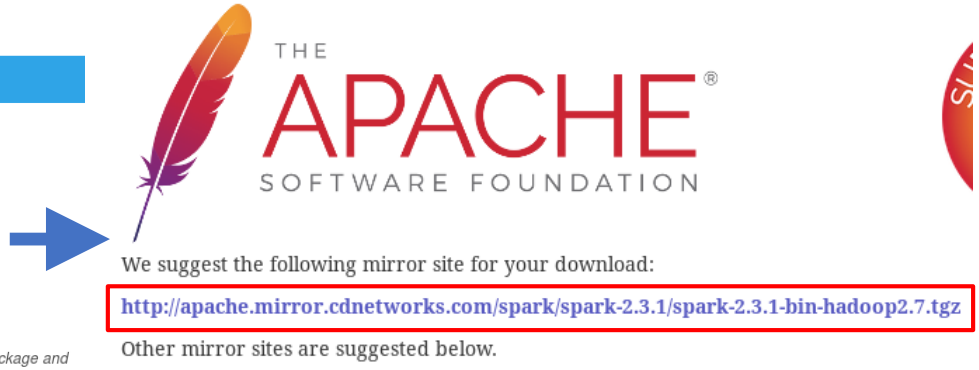
- Apache Spark - v2.3.1 (latest version)

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/
```

Install Apache Spark

1) Spark 홈페이지에서 설치: http://spark.apache.org/downloads.html





2) Command로 설치

```
$ wget http://apache.mirror.cdnetworks.com/spark/spark-2.3.1/spark-2.3.1-bin-hadoop2.7.tgz
```

➔ spark-2.3.1-bin-hadoop2.7.tgz 파일 다운로드

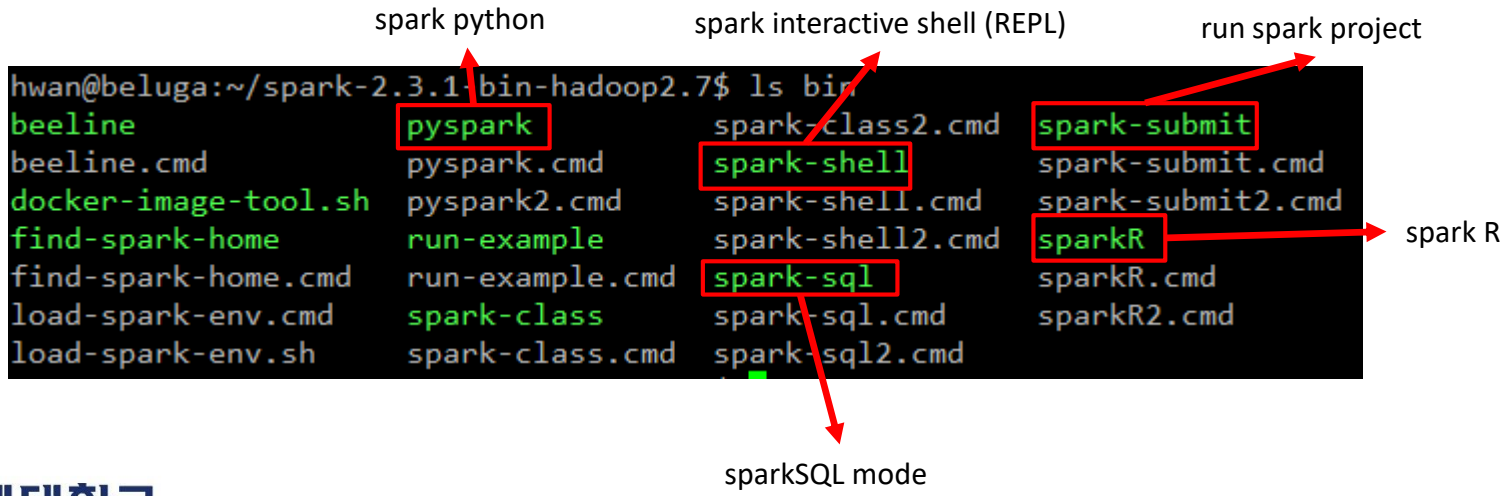
Install Apache Spark

* 압축 풀기

```
$ tar -xvf spark-2.3.1-bin-hadoop2.7.tgz
```

➔ spark-2.3.1-bin-hadoop2.7 디렉토리 생성

* 실행 파일 확인 - spark-2.3.1-bin-hadoop2.7/bin



Install Apache Spark

* 환경 변수(SPARK_HOME) 설정

```
$ sudo mv spark-2.3.1-bin-hadoop2.7 /usr/local/spark
$ vi ~/.bashrc

export SPARK_HOME=/usr/local/spark
export PATH=$SPARK_HOME/bin:$PATH

$ source ~/.bashrc
```

RDD basic operations

* RDD operations
    1) Transformation - RDD를 변환하여 **새로운 RDD 리턴**
        ex) map, filter, flatMap, join, …
    2) Action - RDD를 이용하여 **최종 계산 결과 리턴**
        ex) count, collect, reduce, first, …

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

그림: http://pubdata.tistory.com/38
공식 문서: https://spark.apache.org/docs/latest/rdd-programming-guide.html

Run spark-shell

* spark-shell 실행

```
$ spark-shell
```

```
hwan@beluga:~$ spark-shell
2018-07-03 12:05:06 WARN  Utils:66 - Your hostname, beluga resolves to a loopback address
: 127.0.1.1; using 165.132.106.75 instead (on interface enp5s0f0)
2018-07-03 12:05:06 WARN  Utils:66 - Set SPARK_LOCAL_IP if you need to bind to another ad
dress
2018-07-03 12:05:07 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLeve
l).
Spark context Web UI available at http://165.132.106.75:4040
Spark context available as 'sc' (master = local[*], app id = local-1530587117902).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_171)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

SparkContext

* **SparkContext** ?
  - Spark application의 main entry point
  - SparkConf 객체에 application정보를 담고, 이를 사용하여 SparkContext 생성
  - Spark cluster management
  - **RDD 생성**
  - JVM당 오직 하나의 SparkContext 사용

  - spark-shell 모드에서는 기본적으로 선언되어 있음

```
Spark context Web UI available at http://165.132.106.75:4040
Spark context available as 'sc' (master = local[*], app id = local-1530587117902).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/
```

SparkContext & RDD

* Scala structure(List, Array, …)로부터 RDD 생성

```
val list = sc.parallelize(List(1,2,3,4,7,8,9,10))
```

```
scala> val list = sc.parallelize(List(1,2,3,4,7,8,9,10))
list: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console
>:24
```

* File의 모든 데이터를 String Array로 받아와 RDD 생성

```
val text = sc.textFile("/usr/local/spark/README.md")
```

```
scala> val text = sc.textFile("/usr/local/spark/README.md")
text: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[8] a
t textFile at <console>:24
```

Transformation

* 1) **map** 연산 - list의 모든 element에 대해 +1 연산 수행한 결과를 list2에 저장

```
val list2 = list.map(x => x+1)
```

```
scala> val list2 = list.map(x => x+1)
list2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[9] at map at <console>:25
```

* 2) **filter** 연산 - list의 모든 element 중 짝수 값만 list3에 저장

```
val list3 = list.filter(x => x%2 == 0)
```

```
scala> val list3 = list.filter( x => x%2 ==0)
list3: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[10] at filter at <console>:25
```

➔ Transformation은 **새 RDD**를 리턴함

Action

* 1) collect 연산

```
list.collect
```

```
scala> list.collect
res12: Array[Int] = Array(1, 2, 3, 4, 7, 8, 9, 10)

scala> list2.collect
res13: Array[Int] = Array(2, 3, 4, 5, 8, 9, 10, 11)

scala> list3.collect
res14: Array[Int] = Array(2, 4, 8, 10)
```

* 2) count 연산

```
list.count
```

```
scala> list.count
res0: Long = 8

scala> list2.count
res1: Long = 8

scala> list3.count
res2: Long = 4
```

➔ Action은 **계산 결과**를 리턴함

Example - WordCount

* line단위의 file content를 word 단위로 나누기

```
val word = text.flatMap(x => x.split(“ “))
```

```
scala> val word = text.flatMap(x => x.split(" "))
word: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at flatMap at <console>:25
```

```
scala> word.collect
res12: Array[String] = Array(#, Apache, Spark, "", Spark, is, a, fast, and, general, cluster, computing
, system, for, Big, Data., It, provides, high-level, APIs, in, Scala,, Java,, Python,, and, R,, and, an
, optimized, engine, that, supports, general, computation, graphs, for, data, analysis., It, also, supp
orts, a, rich, set, of, higher-level, tools, including, Spark, SQL, for, SQL, and, DataFrames,, MLlib,
for, machine, learning,, GraphX, for, graph, processing,, and, Spark, Streaming, for, stream, processin
g., "", <http://spark.apache.org/>, "", "", ##, Online, Documentation, "", You, can, find, the, latest,
 Spark, documentation,, including, a, programming, guide,, on, the, [project, web, page](http://spark.a
pache.org/documentation.html)., This, README, file, only, contains, basic, s...
```

Example - WordCount

* 모든 word에 1씩 카운트하기

```
val pair = word.map(x => (x, 1))
```

```
scala> val pair = text.map(x => (x, 1))
pair: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[5] at map at <console>:2
5
```

```
scala> pair.collect
res0: Array[(String, Int)] = Array((#,1), (Apache,1), (Spark,1), ("",1), (Spark,1), (is,1), (a,1), (fas
t,1), (and,1), (general,1), (cluster,1), (computing,1), (system,1), (for,1), (Big,1), (Data.,1), (It,1)
, (provides,1), (high-level,1), (APIs,1), (in,1), (Scala,,1), (Java,,1), (Python,,1), (and,1), (R,,1),
(and,1), (an,1), (optimized,1), (engine,1), (that,1), (supports,1), (general,1), (computation,1), (grap
hs,1), (for,1), (data,1), (analysis.,1), (It,1), (also,1), (supports,1), (a,1), (rich,1), (set,1), (of,
1), (higher-level,1), (tools,1), (including,1), (Spark,1), (SQL,1), (for,1), (SQL,1), (and,1), (DataFra
mes,,1), (MLlib,1), (for,1), (machine,1), (learning,,1), (GraphX,1), (for,1), (graph,1), (processing,,1
), (and,1), (Spark,1), (Streaming,1), (for,1), (stream,1), (processing.,1), ...
```

Example - WordCount

* 같은 word(key)에 대해서 카운팅 집계 하기

```
val result = pair.reduceByKey(_+_)
```

```
scala> val result = pair.reduceByKey(_+_)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[11] at reduceByKey at <cons
ole>:25
```

```
scala> result.collect
res12: Array[(String, Int)] = Array((package,1), (this,1), (Version"](http://spark.apache
.org/docs/latest/building-spark.html#specifying-the-hadoop-version),1), (Because,1), (Pyt
hon,2), (page](http://spark.apache.org/documentation.html).,1), (cluster.,1), (its,1), ([
run,1), (general,3), (have,1), (pre-built,1), (YARN,,1), (locally,2), (changed,1), (local
ly.,1), (sc.parallelize(1,1), (only,1), (several,1), (This,2), (basic,1), (Configuration,
1), (learning,,1), (documentation,3), (first,1), (graph,1), (Hive,2), (info,1), (["Specif
ying,1), ("yarn",1), ([params]`.,1), ([project,1), (prefer,1), (SparkPi,2), (<http://spar
k.apache.org/>,1), (engine,1), (version,1), (file,1), (documentation,,1), (MASTER,1), (ex
ample,3), (["Parallel,1), (are,1), (params,1), (scala>,1), (DataFrames,,1), (provides...
```

* 원하는 word의 필터링 결과 출력 (“Spark”)

```
result.filter(x => x._1 == “Spark”).collect
```

```
scala> result.filter(x => x._1 == "Spark").collect
res14: Array[(String, Int)] = Array((Spark,16))
```

Reference

* SparkConf
    - https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.SparkConf
* SparkContext
    - https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.SparkContext
* RDD operations
    - https://spark.apache.org/docs/latest/rdd-programming-guide.html
* Example
    - https://www.slideshare.net/KangDognhyun/apache-spark-70360736

SparkSQL Demo

연세대학교 컴퓨터과학과 박상현
2018년 7월

Contents

- SparkSQL Language Manual

- Run SparkSQL
  1) Run spark-shell
     SparkSession
     Example
  2) Run spark-sql

Spark SQL Language Manual

This is a complete list of Data Definition Language (DDL) and Data Manipulation Language (DML) constructs supported in Databricks.

- Alter Database
- Alter Table or View
- Alter Table Partitions
- Analyze Table
- Cache
- Cache Table
- Clear Cache
- Create Database
- Create Function
- Create Table
- Create View
- Describe Database
- Describe Function
- Describe Table
- Drop Database
- Drop Function
- Drop Table
- Explain
- Insert
- Load Data
- Refresh Table
- Reset
- Select
- Set
- Show Columns
- Show Create Table
- Show Functions
- Show Partitions
- Show Table Properties
- Truncate Table
- Uncache Table
- Use Database
- Vacuum

Create Table

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [db_name.]table_name
    [(col_name1 col_type1 [COMMENT col_comment1], ...)]
    USING datasource
    [OPTIONS (key1=val1, key2=val2, ...)]
    [PARTITIONED BY (col_name1, col_name2, ...)]
    [CLUSTERED BY (col_name3, col_name4, ...) INTO num_buckets BUCKETS]
    [LOCATION path]
    [COMMENT table_comment]
    [TBLPROPERTIES (key1=val1, key2=val2, ...)]
    [AS select_statement]
```


➔ https://docs.databricks.com/spark/latest/spark-sql/index.html

Run SparkSQL

1) spark (spark-shell)
   - spark 단에서 SQL 관련 클래스들을 import 하고 사용
   - 원하는 SQL을 함수를 통해 사용
     ex) DataFrame.**sql**("SELECT * FROM table1 WHERE @@")
   - RDD <-> Relational 간의 변환이 자유로움

```
$ spark-shell
```

```
scala>
```

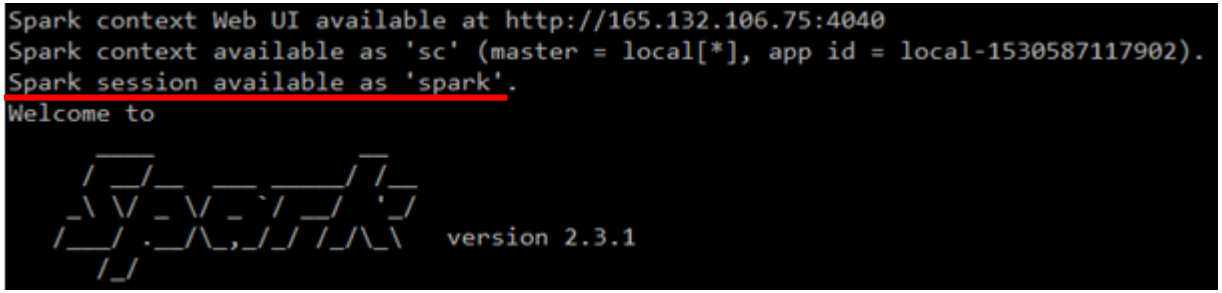
2) spark-sql
   - 오직 SQL을 입력하는 모드로 작동 (일반적인 관계형 데이터베이스 사용처럼)

```
$ spark-sql
```

```
spark-sql>
```

Run spark-shell

* SparkSession
  - DataFrame API를 사용하기 위해 쓰임
  - spark-shell 모드에서는 기본적으로 선언되어 있음



* SparkSession.implicits._
  - RDD <-> DataFrame의 변환에 필요한 함수 포함

```
import spark.implicits._
```

Example

1) json 파일로부터 데이터 읽기 (json, csv, txt, parquet 등 지원)

```
val df = spark.read.json("/usr/local/spark/examples/src/main/
resources/people.json")
```

people.json :

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

```
scala> val df = spark.read.json("/usr/local/spark/examples/src/main/resources/people.json")
df: org.apache.spark.sql.DataFrame = [age: bigint, name: string]
```

2) DataFrame processing

```
scala> df.show
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
scala> df.select($"name").show
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+
```

```
scala> df.where($"age" >20).show
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

```
scala> df.printSchema
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

Example

3) SQL문 실행을 위한 Temporary view 등록

```
df.createOrReplaceTempView(“people”)
```

```
scala> df.createOrReplaceTempView("people")

scala>
```

4) SQL문 직접 실행

```
spark.sql(“SELECT * FROM people WHERE age > 10 ”).show
```

```
scala> spark.sql("SELECT * FROM people WHERE age > 10").show
+---+------+
|age|  name|
+---+------+
| 30|  Andy|
| 19|Justin|
+---+------+
```

5) local db (metastore_db)에 저장

```
df.write.saveAsTable(“people”)
```

➔ spark-shell의 재시작 후 혹은 spark-sql에서도 사용 가능

Run spark-sql

* spark-sql 실행

```
$ spark-sql
```

* table 확인

```
show tables;
```

```
spark-sql> show tables;
18/07/13 17:34:28 INFO SparkSqlParser: Parsing command: show tables
18/07/13 17:34:29 INFO HiveMetaStore: 0: create_database: Database(name:default, description:default da
tabase, locationUri:file:/usr/local/spark/spark-warehouse, parameters:{})
```

```
18/07/13 17:34:33 INFO DAGScheduler: Job 0 finished: processCmd at CliDriver.java:376, took 1.035048 s
people  false
Time taken: 4.866 seconds, Fetched 1 row(s)
18/07/13 17:34:33 INFO CliDriver: Time taken: 4.866 seconds, Fetched 1 row(s)
spark-sql>
```

Run spark-sql

* SELECT문 실행

```
SELECT * FROM people;
```

```
spark-sql> SELECT * FROM people;
18/07/13 17:38:14 INFO SparkSqlParser: Parsing command: SELECT * FROM people
18/07/13 17:38:14 INFO HiveMetaStore: 0: get_table : db=default tbl=people
18/07/13 17:38:14 INFO audit: ugi=hwan  ip=unknown-ip-addr      cmd=get_table : db=default tbl=people
18/07/13 17:38:14 INFO HiveMetaStore: 0: get_table : db=default tbl=people
18/07/13 17:38:14 INFO audit: ugi=hwan  ip=unknown-ip-addr      cmd=get_table : db=default tbl=people
18/07/13 17:38:15 INFO HiveMetaStore: 0: get_table : db=default tbl=people
18/07/13 17:38:15 INFO audit: ugi=hwan  ip=unknown-ip-addr      cmd=get_table : db=default tbl=people
```



```
18/07/13 17:38:16 INFO DAGScheduler: Job 1 finished: processCmd at CliDriver.java:376, took 0.498319 s
NULL    Michael
30      Andy
19      Justin
Time taken: 1.714 seconds, Fetched 3 row(s)
18/07/13 17:38:16 INFO CliDriver: Time taken: 1.714 seconds, Fetched 3 row(s)
spark-sql>
```

Reference

* SparkSQL programming guide

  - https://spark.apache.org/docs/latest/sql-programming-guide.html