Activity Sheet

Learning Outcomes:

Understand how Apache Sqoop efficiently transfers bulk data between Hadoop and structured data stores such as relational databases.

Some of the application areas of Apache Sqoop are.

1. Sqoop helps ETL processing from the EDW to Hadoop for efficient execution at much lower cost.
2. Used to extract data from Hadoop and export into external data structured data stores.
3. Works with all popular databases such as MySQL, MS SQL Server, Oracle, Teradata, Netezza, Postgres and HSQLDB etc.
4. Import sequential datasets from legacy systems such as mainframes.

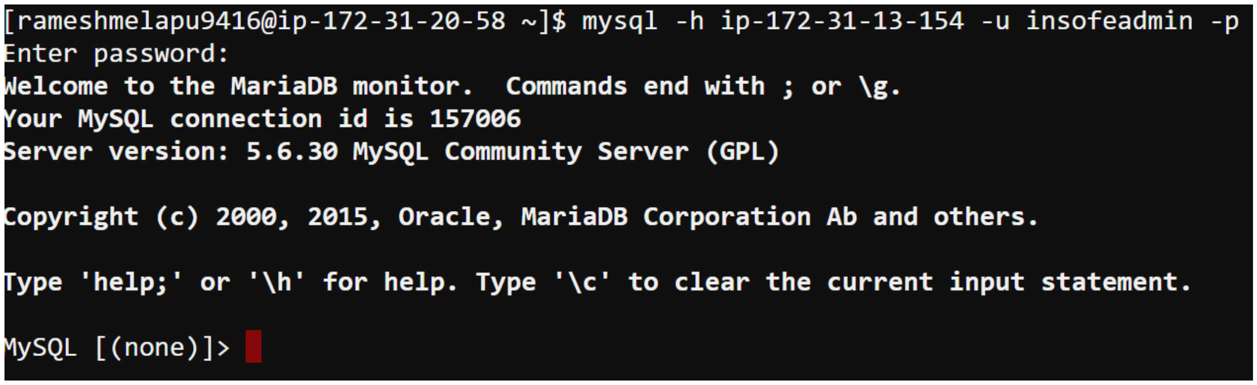
In this example, we will import data from a relational database MySQL to Hive and Process the data using Hive and export the processed data back to MySQL.

Detailed steps are

1. Create a Database and table(s) in MySQL.
2. Load data into table.
3. Import data from MySQL table to Hive using Sqoop Import.
4. Aggregate/Process the data using Hive and Write the results to HDFS.
5. Export above results to MySQL using Sqoop Export.

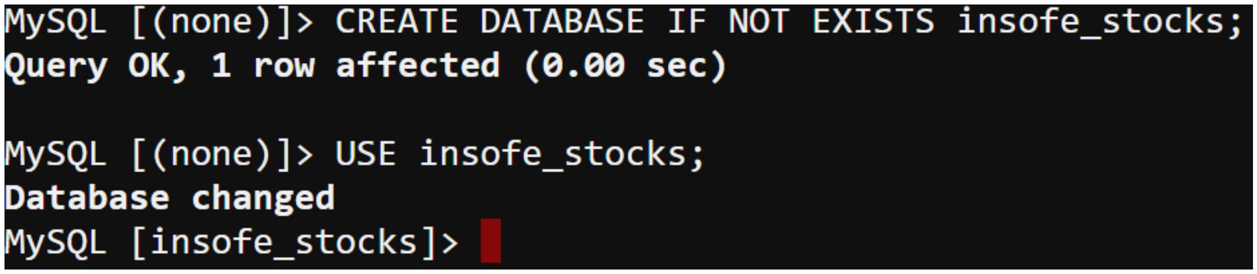
Step1: Create Corresponding Database, Table(s) in MySQL.

a. Access MySQL using the appropriate user credentials.



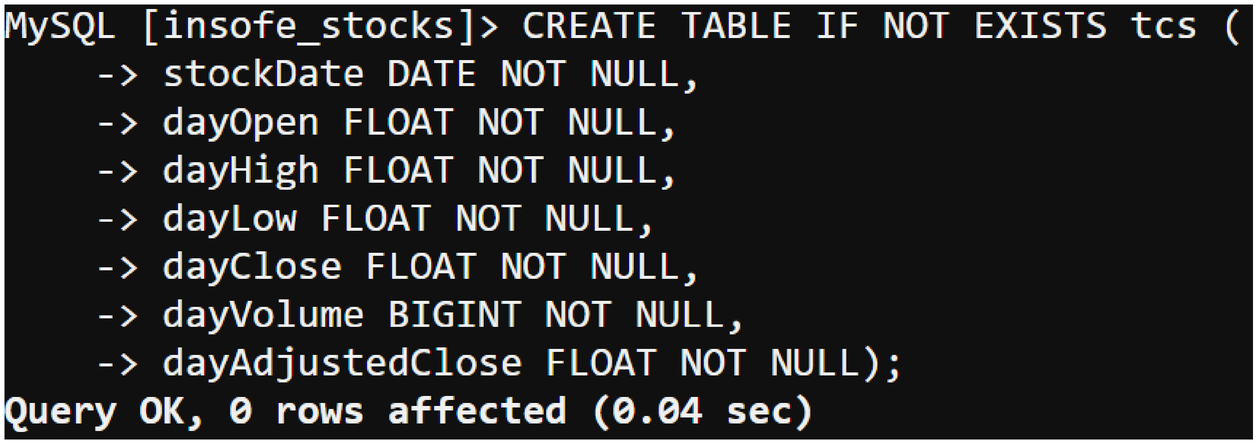
```
[rameshmelapu9416@ip-172-31-20-58 ~]$ mysql -h ip-172-31-13-154 -u insofeadmin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 157006
Server version: 5.6.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

b. Create database in MySQL.



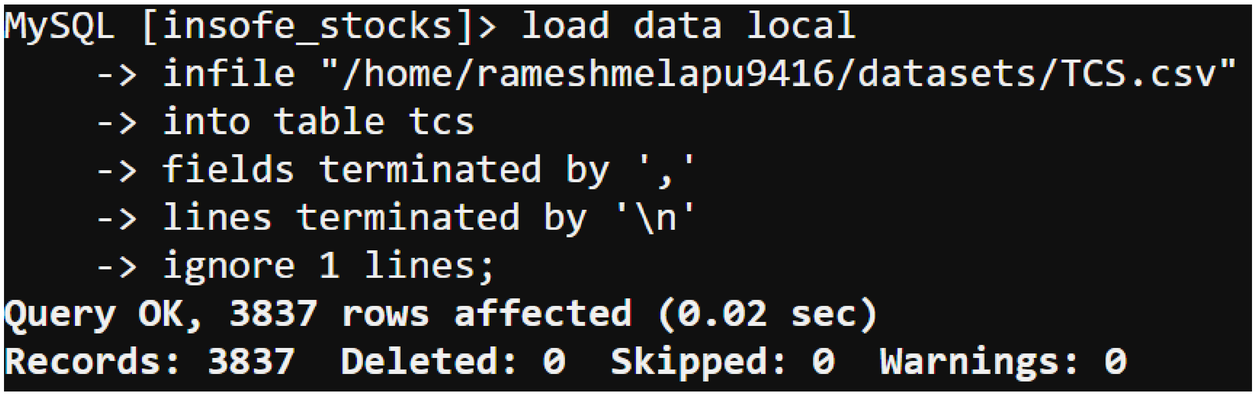
```
MySQL [(none)]> CREATE DATABASE IF NOT EXISTS insofe_stocks;
Query OK, 1 row affected (0.00 sec)

MySQL [(none)]> USE insofe_stocks;
Database changed
MySQL [insofe_stocks]>
```

c. Create table.



```
MySQL [insofe_stocks]> CREATE TABLE IF NOT EXISTS tcs (
    -> stockDate DATE NOT NULL,
    -> dayOpen FLOAT NOT NULL,
    -> dayHigh FLOAT NOT NULL,
    -> dayLow FLOAT NOT NULL,
    -> dayClose FLOAT NOT NULL,
    -> dayVolume BIGINT NOT NULL,
    -> dayAdjustedClose FLOAT NOT NULL);
Query OK, 0 rows affected (0.04 sec)
```

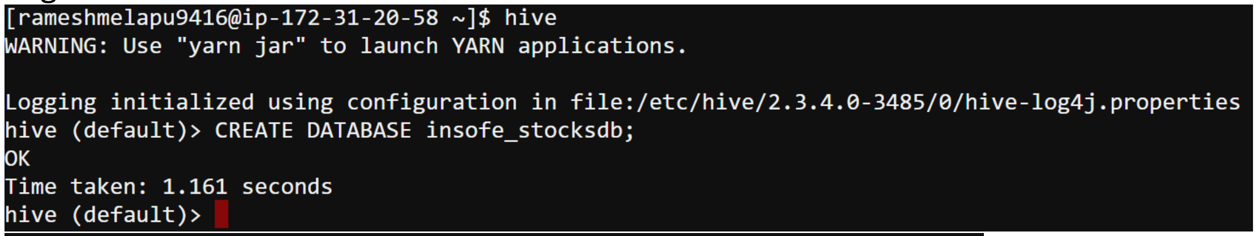
Step2: Load data into the previously created table(s).



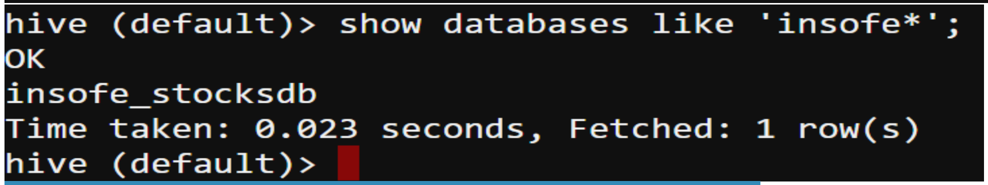
```
MySQL [insofe_stocks]> load data local
    -> infile "/home/rameshmelapu9416/datasets/TCS.csv"
    -> into table tcs
    -> fields terminated by ','
    -> lines terminated by '\n'
    -> ignore 1 lines;
Query OK, 3837 rows affected (0.02 sec)
Records: 3837  Deleted: 0  Skipped: 0  Warnings: 0
```

Step3: Import data from MySQL to Hive using Sqoop.

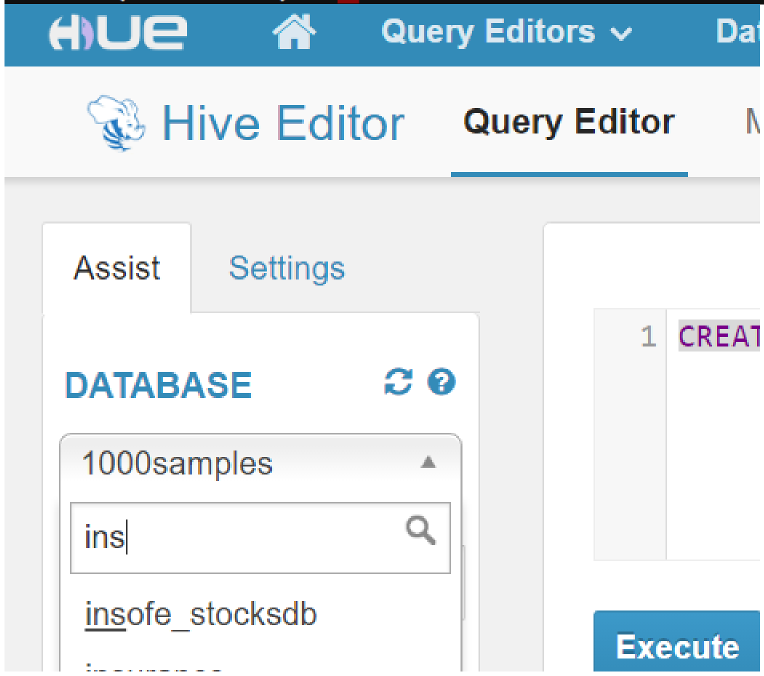
a. Create database in Hive – to import tables to this specific database.
   Login to either hive CLI or HUE.



```
[rameshmelapu9416@ip-172-31-20-58 ~]$ hive
WARNING: Use "yarn jar" to launch YARN applications.

Logging initialized using configuration in file:/etc/hive/2.3.4.0-3485/0/hive-log4j.properties
hive (default)> CREATE DATABASE insofe_stocksdb;
OK
Time taken: 1.161 seconds
hive (default)>
```



```
hive (default)> show databases like 'insofe*';
OK
insofe_stocksdb
Time taken: 0.023 seconds, Fetched: 1 row(s)
hive (default)>
```

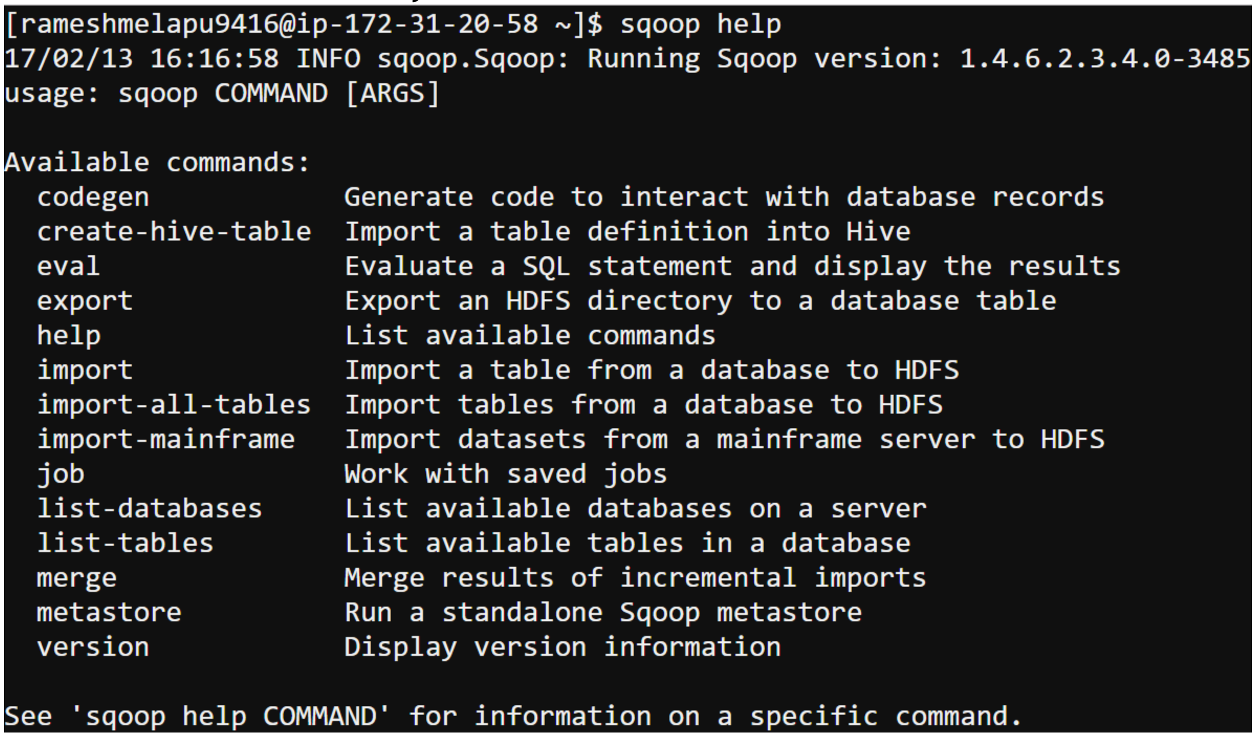


b. Import data into existing hive database using Sqoop.

Sqoop

Apache Sqoop is an open source tool that allows users to extract data from a structured data store into Hadoop for further processing. This processing can be done with MapReduce programs or other higher-level tools such as Hive. (It’s even possible to use Sqoop to move data from a database into HBase.) When the final results of an analytic pipeline are available, Sqoop can export these results back to the data store for consumption by other clients.

Sqoop is organized as a set of tools or commands. It can print out the list of available tools, like this:



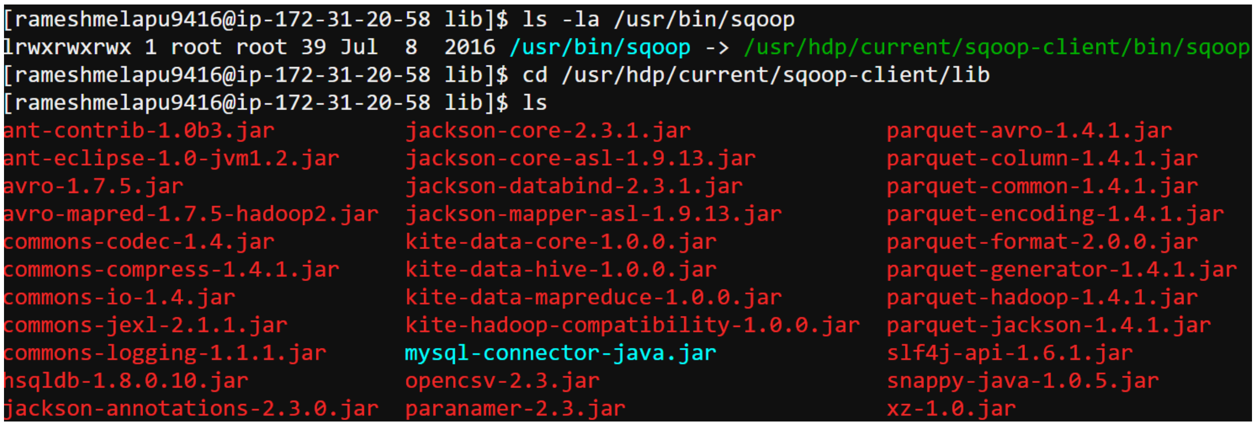
```
[rameshmelapu9416@ip-172-31-20-58 ~]$ sqoop help
17/02/13 16:16:58 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6.2.3.4.0-3485
usage: sqoop COMMAND [ARGS]

Available commands:
  codegen            Generate code to interact with database records
  create-hive-table  Import a table definition into Hive
  eval               Evaluate a SQL statement and display the results
  export             Export an HDFS directory to a database table
  help               List available commands
  import             Import a table from a database to HDFS
  import-all-tables  Import tables from a database to HDFS
  import-mainframe   Import datasets from a mainframe server to HDFS
  job                Work with saved jobs
  list-databases     List available databases on a server
  list-tables        List available tables in a database
  merge              Merge results of incremental imports
  metastore          Run a standalone Sqoop metastore
  version            Display version information

See 'sqoop help COMMAND' for information on a specific command.
```

An alternate way of running a Sqoop tool is to use a tool-specific script. This script will be named *sqoop-toolname* (e.g., *sqoop-help*, *sqoop-import*, etc.). Running these scripts from the command line is identical to running sqoop help or sqoop import.
Ex. $ sqoop-help

Sqoop Connectors

Sqoop has an extension framework that makes it possible to import data from—and export data to—any external storage system that has bulk data transfer capabilities. A Sqoop *connector* is a modular component that uses this framework to enable Sqoop imports and exports. Sqoop ships with connectors for working with a range of popular databases, including MySQL, PostgreSQL, Oracle, SQL Server, DB2, and Netezza. There is also a generic JDBC connector for connecting to any database that supports Java's JDBC protocol. Sqoop provides optimized MySQL, PostgreSQL, Oracle, and Netezza connectors that use database-specific APIs to perform bulk transfers more efficiently.

As well as the built-in Sqoop connectors, various third-party connectors are available for data stores, ranging from enterprise data warehouses (such as Teradata) to NoSQL stores (such as Couchbase). These connectors must be downloaded separately and can be added to an existing Sqoop installation by following the instructions that come with the connector.



```
[rameshmelapu9416@ip-172-31-20-58 lib]$ ls -la /usr/bin/sqoop
lrwxrwxrwx 1 root root 39 Jul  8  2016 /usr/bin/sqoop -> /usr/hdp/current/sqoop-client/bin/sqoop
[rameshmelapu9416@ip-172-31-20-58 lib]$ cd /usr/hdp/current/sqoop-client/lib
[rameshmelapu9416@ip-172-31-20-58 lib]$ ls
ant-contrib-1.0b3.jar          jackson-core-2.3.1.jar                parquet-avro-1.4.1.jar
ant-eclipse-1.0-jvm1.2.jar     jackson-core-asl-1.9.13.jar           parquet-column-1.4.1.jar
avro-1.7.5.jar                 jackson-databind-2.3.1.jar            parquet-common-1.4.1.jar
avro-mapred-1.7.5-hadoop2.jar  jackson-mapper-asl-1.9.13.jar         parquet-encoding-1.4.1.jar
commons-codec-1.4.jar          kite-data-core-1.0.0.jar              parquet-format-2.0.0.jar
commons-compress-1.4.1.jar     kite-data-hive-1.0.0.jar              parquet-generator-1.4.1.jar
commons-io-1.4.jar             kite-data-mapreduce-1.0.0.jar         parquet-hadoop-1.4.1.jar
commons-jexl-2.1.1.jar         kite-hadoop-compatibility-1.0.0.jar   parquet-jackson-1.4.1.jar
commons-logging-1.1.1.jar      mysql-connector-java.jar              slf4j-api-1.6.1.jar
hsqldb-1.8.0.10.jar            opencsv-2.3.jar                       snappy-java-1.0.5.jar
jackson-annotations-2.3.0.jar  paranamer-2.3.jar                     xz-1.0.jar
```

Basic Sqoop Commands:

```
$ sqoop list-databases --connect jdbc:mysql://localhost --username root
$ sqoop list-tables --connect jdbc:mysql://localhost/test --username  root
```

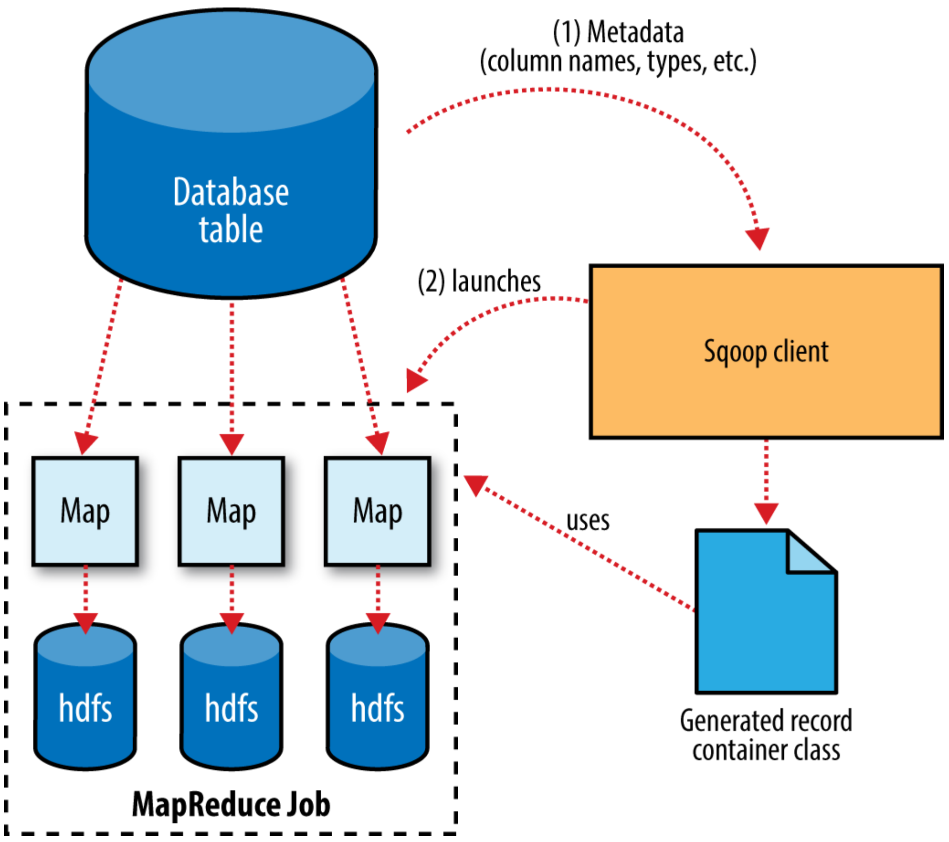
Sqoop Import

In this listing, we created a new table called users. The users table contains several fields representing a variety of data types.

Before going any further, you need to download the JDBC driver JAR file for MySQL (Connector/J) and add it to Sqoop's classpath, which is simply achieved by placing it in Sqoop's */lib* directory. Now let's use Sqoop to import this table into HDFS:

```
$ sqoop import \
--connect jdbc:mysql://localhost/test \
--table users \
--username root \
-m 1 \
--target-dir /tmp/sqoopOut/sql/1

$ sqoop import \
--connect jdbc:mysql://localhost/test \
--table users \
--fields-terminated-by '\t' \
--where "age > 25" \
--username root -m 1 \
--target-dir /tmp/sqoopOut/sql/2
```

Sqoop's import tool will run a MapReduce job that connects to the MySQL database and reads the table. By default, this will use four map tasks in parallel to speed up the import process. Each task will write its imported results to a different file, but all in a common directory. Because we knew that we had about 900 rows to import in this example, we specified that Sqoop should use a single map task (-m 1) so we get a single file in HDFS.



The connect string (jdbc:mysql://localhost/test) shown in the example will read from a database on the local machine. If a distributed Hadoop cluster is being used, localhost should not be specified in the connect string, because map tasks not running on the same machine as the database will fail to connect. Even if Sqoop is run from the same host as the database server, the full hostname should be specified.

By default, Sqoop will generate comma-delimited text files for our imported data. Delimiters can be specified explicitly, as well as field enclosing and escape characters, to allow the presence of delimiters in the field contents. The command-line arguments that specify delimiter characters, file formats, compression, and more fine-grained control of the import process are described in the Sqoop User Guide distributed with Sqoop, as well as in the online help (sqoop help import, or man sqoop-import in CDH).

Generated Code

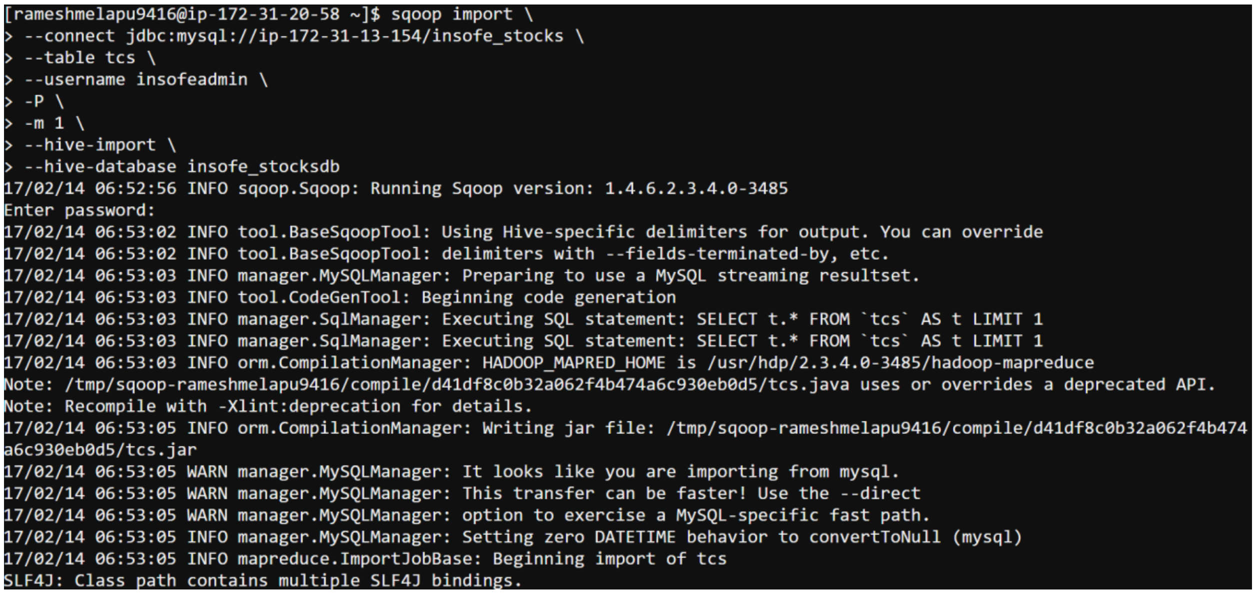
In addition to writing the contents of the database table to HDFS, Sqoop also provides with a generated Java source file (*users.java*) written to the current local directory.
(After running the sqoop import command shown earlier, you can see this file by running ls users.java.)
Sqoop can use generated code to handle the deserialization of table-specific data from the database source before writing it to HDFS.

```
$ sqoop codegen --connect jdbc:mysql://localhost/test \
  --table users --class-name Users
```

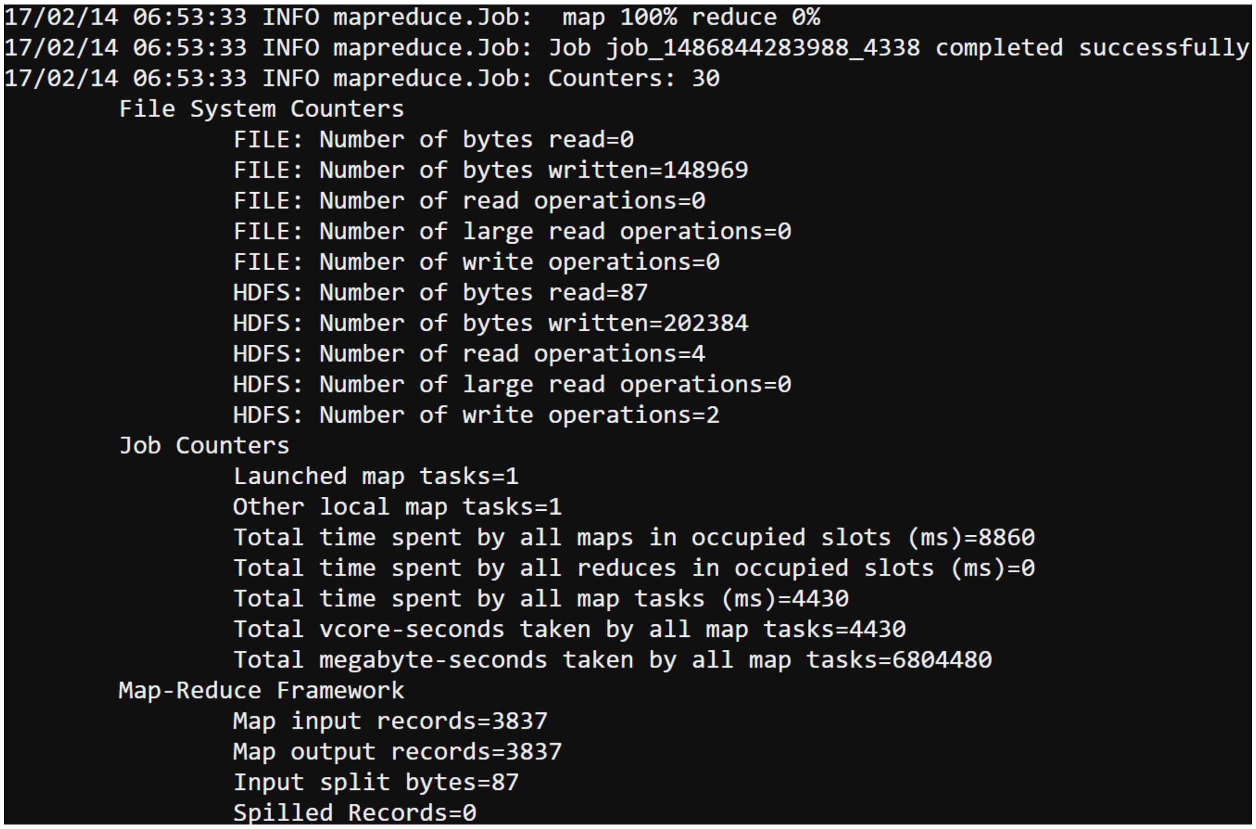
Based on the URL in the connect string used to access the database, Sqoop attempts to predict which driver it should load. You still need to download the JDBC driver itself and install it on your Sqoop client. For cases where Sqoop does not know which JDBC driver is appropriate, users can specify the JDBC driver explicitly with the --driver argument. This capability allows Sqoop to work with a wide variety of database platforms.

Before the import can start, Sqoop uses JDBC to examine the table it is to import. It retrieves a list of all the columns and their SQL data types. These SQL types (VARCHAR, INTEGER, etc.) can then be mapped to Java data types (String, Integer, etc.), which will hold the field values in MapReduce applications. Sqoop's code generator will use this information to create a table-specific class to hold a record extracted from the table.

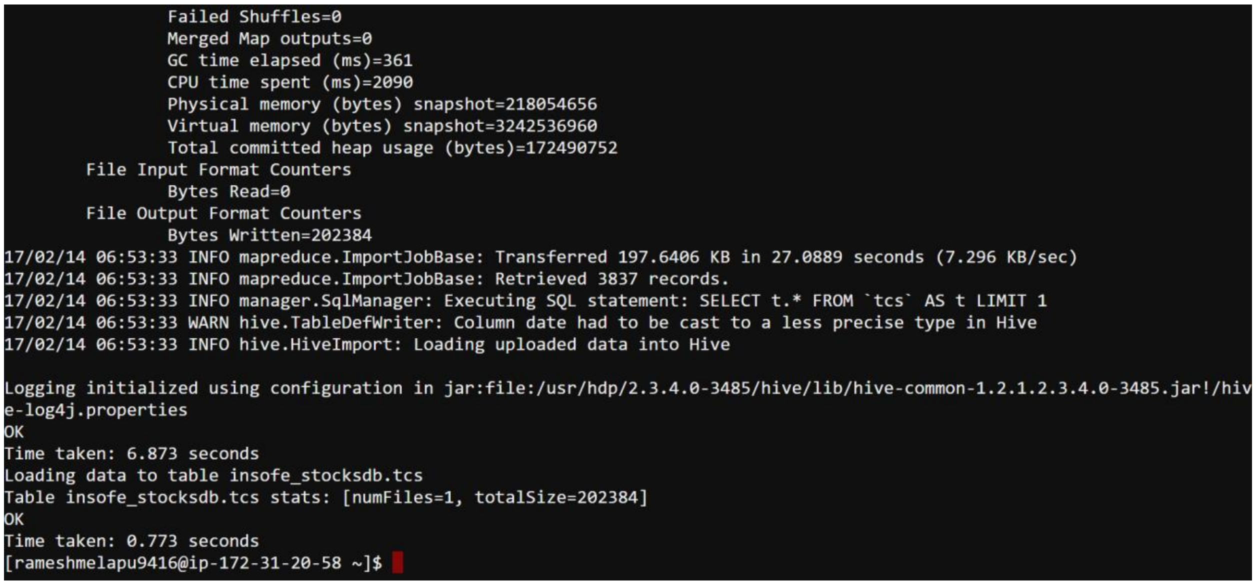
Import data into existing hive database using sqoop.
Verify Job Counters, Job Status:



```
[rameshmelapu9416@ip-172-31-20-58 ~]$ sqoop import \
> --connect jdbc:mysql://ip-172-31-13-154/insofe_stocks \
> --table tcs \
> --username insofeadmin \
> -P \
> -m 1 \
> --hive-import \
> --hive-database insofe_stocksdb
17/02/14 06:52:56 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6.2.3.4.0-3485
Enter password:
17/02/14 06:53:02 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
17/02/14 06:53:02 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
17/02/14 06:53:03 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
17/02/14 06:53:03 INFO tool.CodeGenTool: Beginning code generation
17/02/14 06:53:03 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tcs` AS t LIMIT 1
17/02/14 06:53:03 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tcs` AS t LIMIT 1
17/02/14 06:53:03 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/hdp/2.3.4.0-3485/hadoop-mapreduce
Note: /tmp/sqoop-rameshmelapu9416/compile/d41df8c0b32a062f4b474a6c930eb0d5/tcs.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
17/02/14 06:53:05 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-rameshmelapu9416/compile/d41df8c0b32a062f4b474
a6c930eb0d5/tcs.jar
17/02/14 06:53:05 WARN manager.MySQLManager: It looks like you are importing from mysql.
17/02/14 06:53:05 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
17/02/14 06:53:05 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
17/02/14 06:53:05 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
17/02/14 06:53:05 INFO mapreduce.ImportJobBase: Beginning import of tcs
SLF4J: Class path contains multiple SLF4J bindings.
```

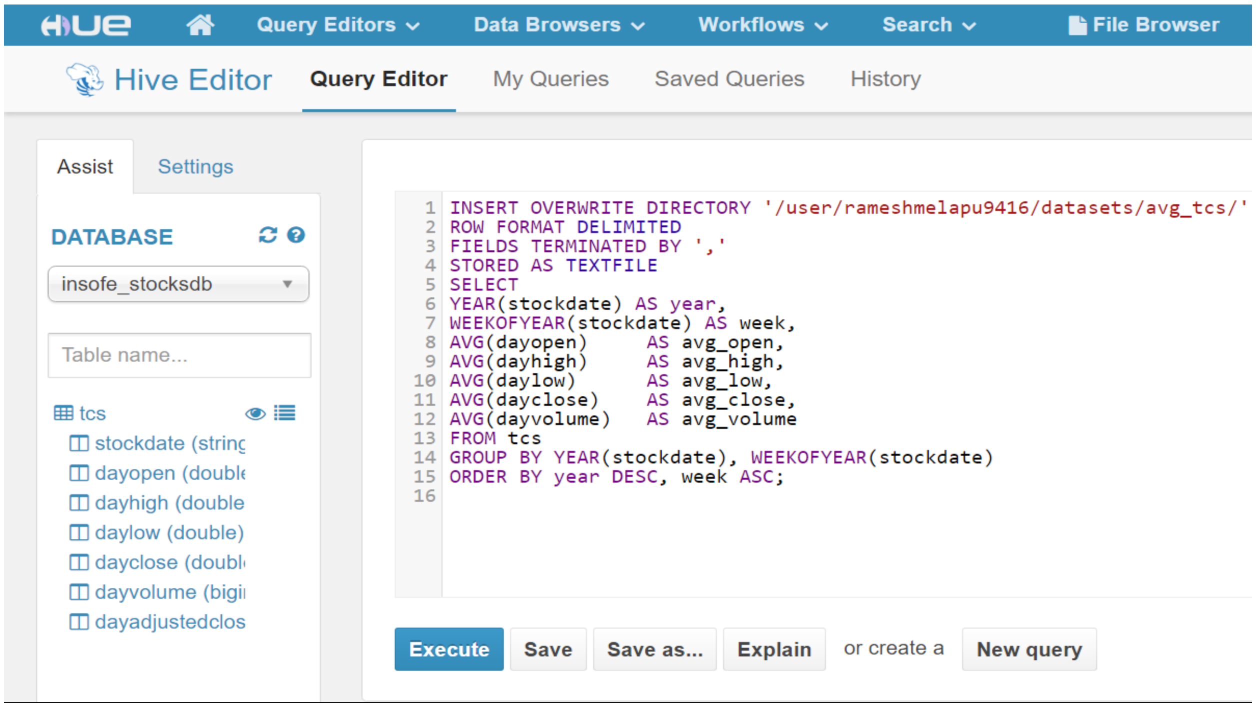


```
17/02/14 06:53:33 INFO mapreduce.Job:  map 100% reduce 0%
17/02/14 06:53:33 INFO mapreduce.Job: Job job_1486844283988_4338 completed successfully
17/02/14 06:53:33 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=148969
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=87
                HDFS: Number of bytes written=202384
                HDFS: Number of read operations=4
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=8860
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=4430
                Total vcore-seconds taken by all map tasks=4430
                Total megabyte-seconds taken by all map tasks=6804480
        Map-Reduce Framework
                Map input records=3837
                Map output records=3837
                Input split bytes=87
                Spilled Records=0
```

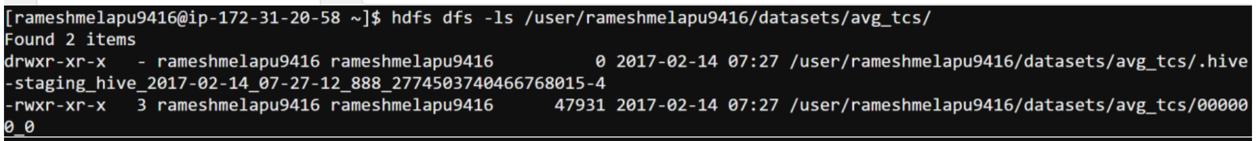


```
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=361
                CPU time spent (ms)=2090
                Physical memory (bytes) snapshot=218054656
                Virtual memory (bytes) snapshot=3242536960
                Total committed heap usage (bytes)=172490752
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=202384
17/02/14 06:53:33 INFO mapreduce.ImportJobBase: Transferred 197.6406 KB in 27.0889 seconds (7.296 KB/sec)
17/02/14 06:53:33 INFO mapreduce.ImportJobBase: Retrieved 3837 records.
17/02/14 06:53:33 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tcs` AS t LIMIT 1
17/02/14 06:53:33 WARN hive.TableDefWriter: Column date had to be cast to a less precise type in Hive
17/02/14 06:53:33 INFO hive.HiveImport: Loading uploaded data into Hive

Logging initialized using configuration in jar:file:/usr/hdp/2.3.4.0-3485/hive/lib/hive-common-1.2.1.2.3.4.0-3485.jar!/hiv
e-log4j.properties
OK
Time taken: 6.873 seconds
Loading data to table insofe_stocksdb.tcs
Table insofe_stocksdb.tcs stats: [numFiles=1, totalSize=202384]
OK
Time taken: 0.773 seconds
[rameshmelapu9416@ip-172-31-20-58 ~]$
```

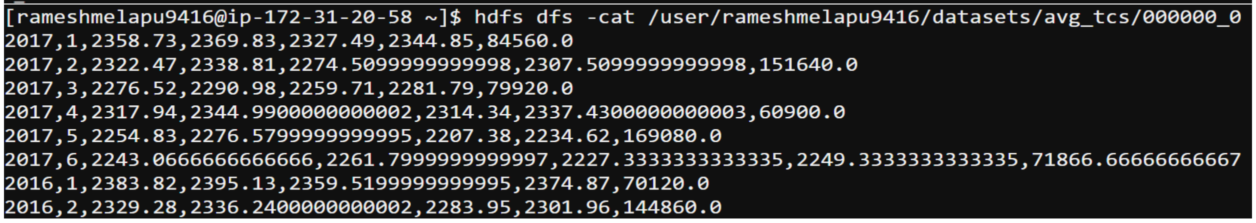
Step4: Aggregate/Process the data using Hive and Write the results to HDFS.



```
INSERT OVERWRITE DIRECTORY '/user/rameshmelapu9416/datasets/avg_tcs/'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
SELECT
YEAR(stockdate) AS year,
WEEKOFYEAR(stockdate) AS week,
AVG(dayopen)      AS avg_open,
AVG(dayhigh)      AS avg_high,
AVG(daylow)       AS avg_low,
AVG(dayclose)     AS avg_close,
AVG(dayvolume)    AS avg_volume
FROM tcs
GROUP BY YEAR(stockdate), WEEKOFYEAR(stockdate)
ORDER BY year DESC, week ASC;
```



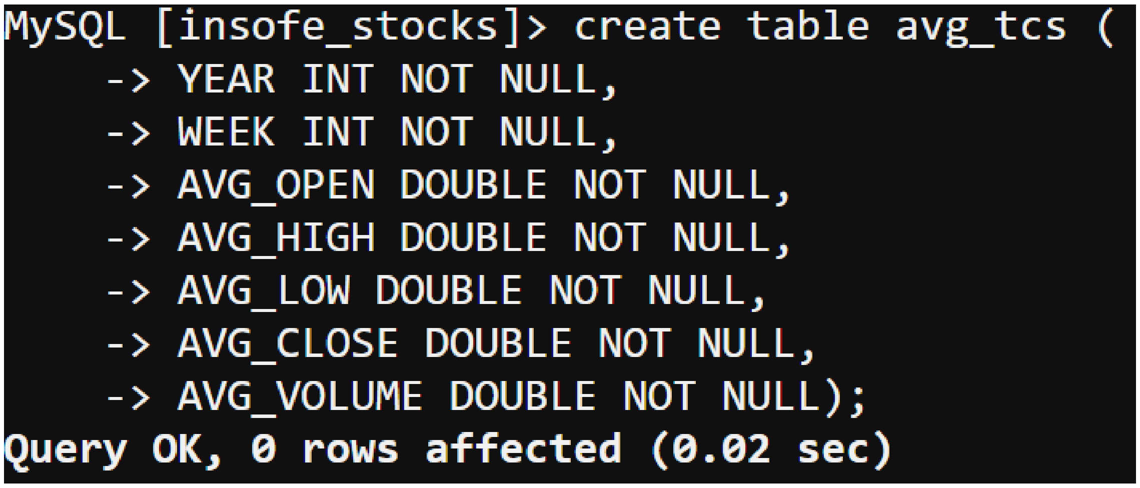
```
[rameshmelapu9416@ip-172-31-20-58 ~]$ hdfs dfs -ls /user/rameshmelapu9416/datasets/avg_tcs/
Found 2 items
drwxr-xr-x   - rameshmelapu9416 rameshmelapu9416          0 2017-02-14 07:27 /user/rameshmelapu9416/datasets/avg_tcs/.hive
-staging_hive_2017-02-14_07-27-12_888_2774503740466768015-4
-rwxr-xr-x   3 rameshmelapu9416 rameshmelapu9416      47931 2017-02-14 07:27 /user/rameshmelapu9416/datasets/avg_tcs/00000
0_0
```



```
[rameshmelapu9416@ip-172-31-20-58 ~]$ hdfs dfs -cat /user/rameshmelapu9416/datasets/avg_tcs/000000_0
2017,1,2358.73,2369.83,2327.49,2344.85,84560.0
2017,2,2322.47,2338.81,2274.5099999999998,2307.5099999999998,151640.0
2017,3,2276.52,2290.98,2259.71,2281.79,79920.0
2017,4,2317.94,2344.9900000000002,2314.34,2337.4300000000003,60900.0
2017,5,2254.83,2276.5799999999995,2207.38,2234.62,169080.0
2017,6,2243.0666666666666,2261.7999999999997,2227.3333333333335,2249.3333333333335,71866.66666666667
2016,1,2383.82,2395.13,2359.5199999999995,2374.87,70120.0
2016,2,2329.28,2336.2400000000002,2283.95,2301.96,144860.0
```

Step5: Export above results to MySQL using Sqoop Export.

a. Create the table in MySQL to store the results.



```
MySQL [insofe_stocks]> create table avg_tcs (
    -> YEAR INT NOT NULL,
    -> WEEK INT NOT NULL,
    -> AVG_OPEN DOUBLE NOT NULL,
    -> AVG_HIGH DOUBLE NOT NULL,
    -> AVG_LOW DOUBLE NOT NULL,
    -> AVG_CLOSE DOUBLE NOT NULL,
    -> AVG_VOLUME DOUBLE NOT NULL);
Query OK, 0 rows affected (0.02 sec)
```

b. Export results (data in HDFS) into above table using Sqoop Export.

Sqoop Export

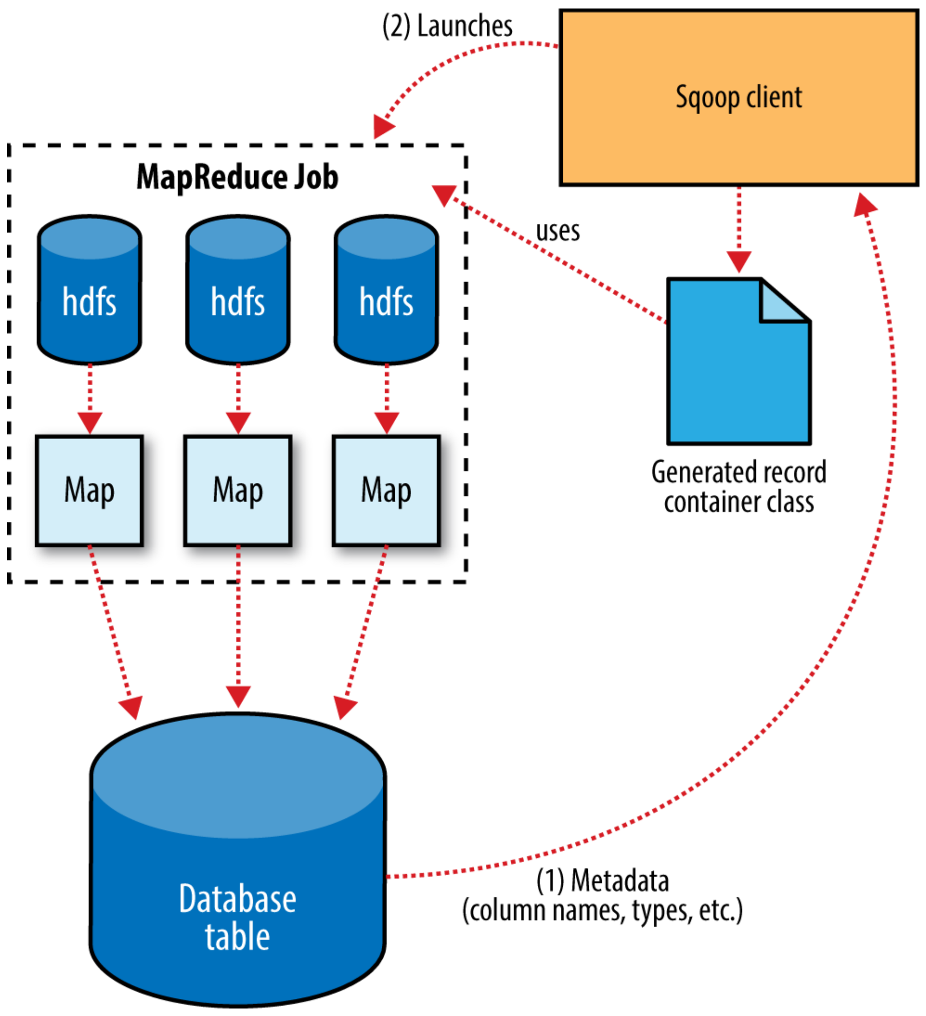
In Sqoop, an *import* refers to the movement of data from a database system into HDFS. By contrast, an *export* uses HDFS as the source of data and a remote database as the destination. In the previous sections, we imported some data and then performed some analysis using Hive. We can export the results of this analysis to a database for consumption by other tools. Before exporting a table from HDFS to a database, we must prepare the database to receive the data by creating the target table. Although Sqoop can infer which Java types are appropriate to hold SQL data types, this translation does not work in both directions (for example, there are several possible SQL column definitions that can hold data in a Java String; this could be CHAR(64), VARCHAR(200), or something else entirely). Consequently, one must determine which types are most appropriate.

Export the users table from HDFS. We need to create a table in MySQL that has target columns in the same order, with the appropriate SQL types:

```
$ sqoop export \
--connect jdbc:mysql://localhost/test \
--table users_new \
--export-dir /tmp/sqoopOut/sql/1 \
--fields-terminated-by '\t' \
--username root
```

The Sqoop performs exports is very similar in nature to how Sqoop performs imports Before performing the export, Sqoop picks a strategy based on the database connect string. For most systems, Sqoop uses JDBC. Sqoop then generates a Java class based on the target table definition. This generated class has the ability to parse records from text files and insert values of the appropriate types into a table (in addition to the ability to read the columns from a ResultSet). A MapReduce job is then launched that reads the source datafiles from HDFS, parses the records using the generated class, and executes the chosen export strategy.

The JDBC-based export strategy builds up batch INSERT statements that will each add multiple records to the target table. Inserting many records per statement performs much better than executing many single-row INSERT statements on most database systems.

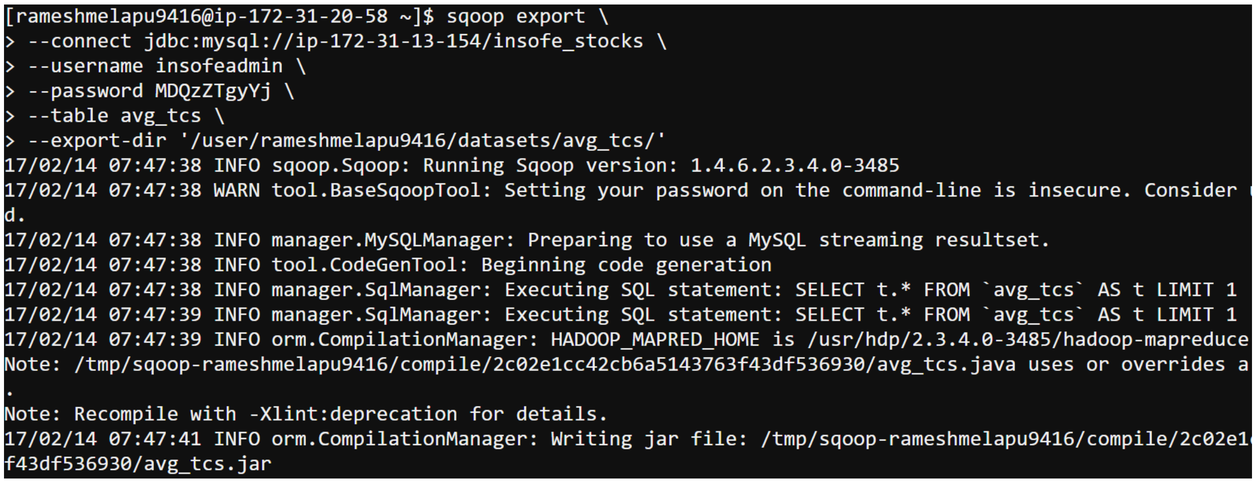


Export the users table from HIVE. We need to create a table in MySQL that has target columns in the same order, with the appropriate SQL types:

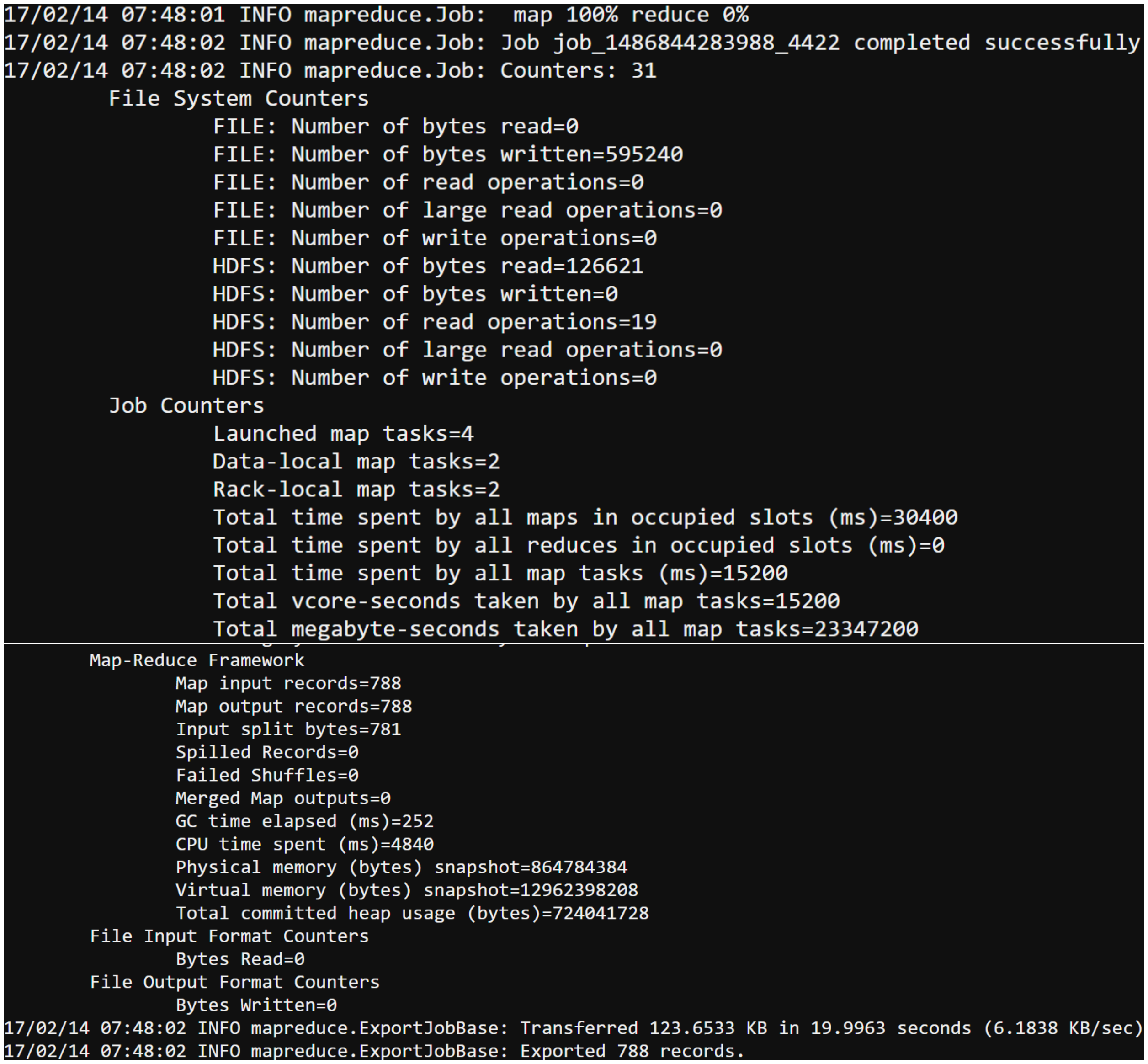
```
$ sqoop export \
--connect jdbc:mysql://localhost/test \
--table users_new \
--export-dir /user/hive/warehouse/users \
--input-fields-terminated-by '\001' \
--username root
```

When we created the users table in Hive, we did not specify any delimiters. So Hive used its default delimiters: a Ctrl-A character (Unicode 0x0001) between fields and a newline at the end of each record. When we used Hive to access the contents of this table (in a SELECT statement), Hive converted this to a tab-delimited representation for display on the console. But when reading the tables directly from files, we need to tell Sqoop which delimiters to use. Sqoop assumes records are newline-delimited by default, but needs to be told about the Ctrl-A field delimiters. The *--input-fieldsterminated-by* argument to sqoop export specified this information. Sqoop supports several escape sequences, which start with a backslash (\) character, when specifying delimiters.
In the example syntax, the escape sequence is enclosed in single quotes to ensure that the shell processes it literally. Without the quotes, the leading backslash itself may need to be escaped (e.g., --input-fields-terminated-by \001).

Export above results to MySQL using Sqoop Export.
Verify Job Counters, Job Status:

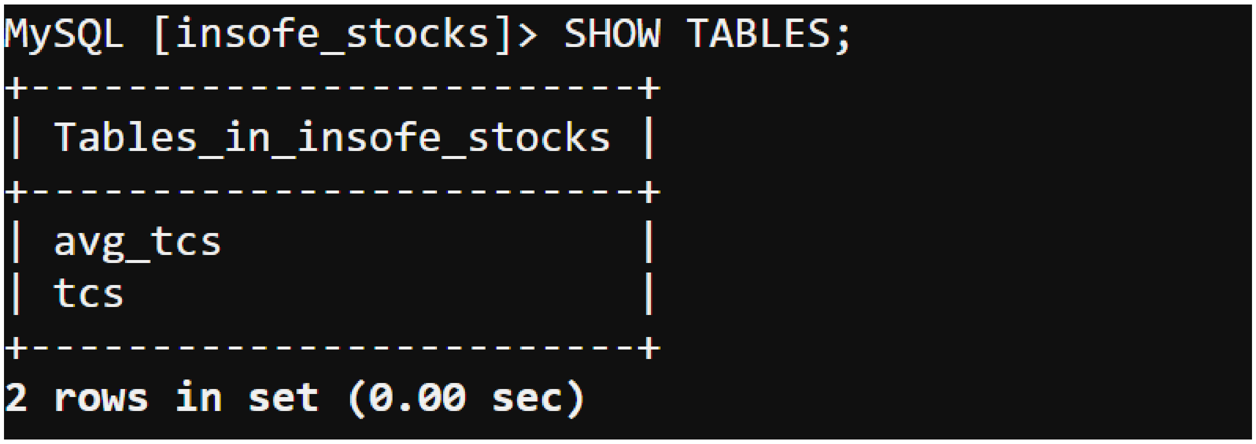


```
[rameshmelapu9416@ip-172-31-20-58 ~]$ sqoop export \
> --connect jdbc:mysql://ip-172-31-13-154/insofe_stocks \
> --username insofeadmin \
> --password MDQzZTgyYj \
> --table avg_tcs \
> --export-dir '/user/rameshmelapu9416/datasets/avg_tcs/'
17/02/14 07:47:38 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6.2.3.4.0-3485
17/02/14 07:47:38 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider
d.
17/02/14 07:47:38 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
17/02/14 07:47:38 INFO tool.CodeGenTool: Beginning code generation
17/02/14 07:47:38 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `avg_tcs` AS t LIMIT 1
17/02/14 07:47:39 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `avg_tcs` AS t LIMIT 1
17/02/14 07:47:39 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/hdp/2.3.4.0-3485/hadoop-mapreduce
Note: /tmp/sqoop-rameshmelapu9416/compile/2c02e1cc42cb6a5143763f43df536930/avg_tcs.java uses or overrides a
.
Note: Recompile with -Xlint:deprecation for details.
17/02/14 07:47:41 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-rameshmelapu9416/compile/2c02e1
f43df536930/avg_tcs.jar
```

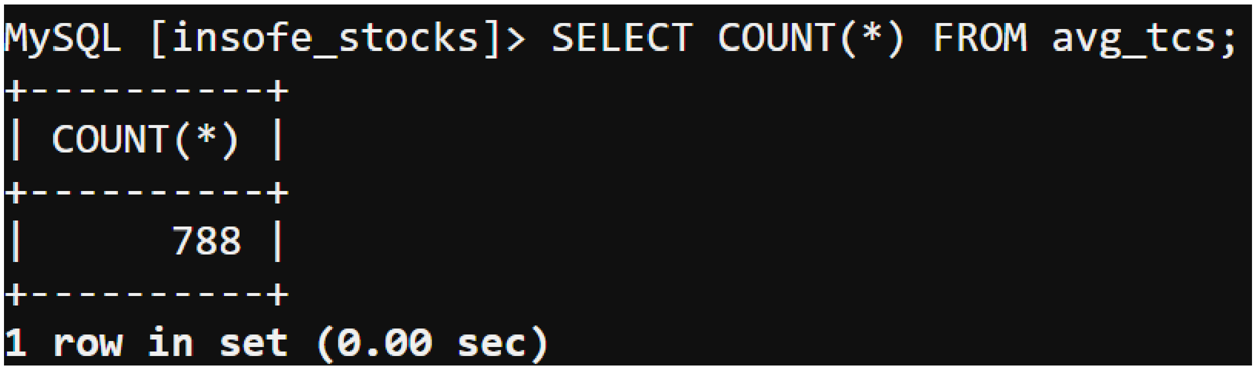


```
17/02/14 07:48:01 INFO mapreduce.Job:  map 100% reduce 0%
17/02/14 07:48:02 INFO mapreduce.Job: Job job_1486844283988_4422 completed successfully
17/02/14 07:48:02 INFO mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=595240
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=126621
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=19
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
        Job Counters
                Launched map tasks=4
                Data-local map tasks=2
                Rack-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=30400
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=15200
                Total vcore-seconds taken by all map tasks=15200
                Total megabyte-seconds taken by all map tasks=23347200
        Map-Reduce Framework
                Map input records=788
                Map output records=788
                Input split bytes=781
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=252
                CPU time spent (ms)=4840
                Physical memory (bytes) snapshot=864784384
                Virtual memory (bytes) snapshot=12962398208
                Total committed heap usage (bytes)=724041728
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=0
17/02/14 07:48:02 INFO mapreduce.ExportJobBase: Transferred 123.6533 KB in 19.9963 seconds (6.1838 KB/sec)
17/02/14 07:48:02 INFO mapreduce.ExportJobBase: Exported 788 records.
```

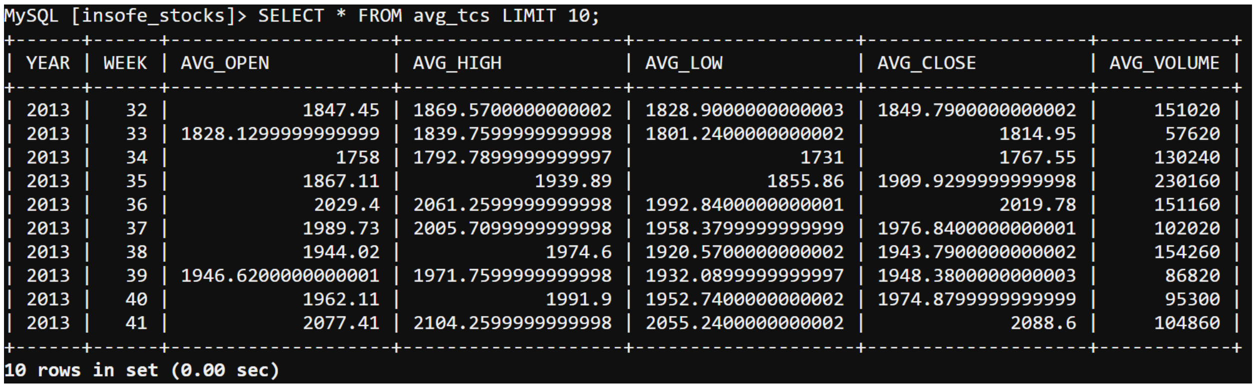
c. Verify results in MySQL



```
MySQL [insofe_stocks]> SHOW TABLES;
+-------------------------+
| Tables_in_insofe_stocks |
+-------------------------+
| avg_tcs                 |
| tcs                     |
+-------------------------+
2 rows in set (0.00 sec)
```



```
MySQL [insofe_stocks]> SELECT COUNT(*) FROM avg_tcs;
+----------+
| COUNT(*) |
+----------+
|      788 |
+----------+
1 row in set (0.00 sec)
```



```
MySQL [insofe_stocks]> SELECT * FROM avg_tcs LIMIT 10;
+------+------+--------------------+--------------------+--------------------+--------------------+------------+
| YEAR | WEEK | AVG_OPEN           | AVG_HIGH           | AVG_LOW            | AVG_CLOSE          | AVG_VOLUME |
+------+------+--------------------+--------------------+--------------------+--------------------+------------+
| 2013 |   32 |            1847.45 | 1869.5700000000002 | 1828.9000000000003 | 1849.7900000000002 |     151020 |
| 2013 |   33 | 1828.1299999999999 | 1839.7599999999998 | 1801.2400000000002 |            1814.95 |      57620 |
| 2013 |   34 |               1758 | 1792.7899999999997 |               1731 |            1767.55 |     130240 |
| 2013 |   35 |            1867.11 |            1939.89 |            1855.86 | 1909.9299999999998 |     230160 |
| 2013 |   36 |             2029.4 | 2061.2599999999998 | 1992.8400000000001 |            2019.78 |     151160 |
| 2013 |   37 |            1989.73 | 2005.7099999999998 | 1958.3799999999999 | 1976.8400000000001 |     102020 |
| 2013 |   38 |            1944.02 |             1974.6 | 1920.5700000000002 | 1943.7900000000002 |     154260 |
| 2013 |   39 | 1946.6200000000001 | 1971.7599999999998 | 1932.0899999999997 | 1948.3800000000003 |      86820 |
| 2013 |   40 |            1962.11 |             1991.9 | 1952.7400000000002 | 1974.8799999999999 |      95300 |
| 2013 |   41 |            2077.41 | 2104.2599999999998 | 2055.2400000000002 |             2088.6 |     104860 |
+------+------+--------------------+--------------------+--------------------+--------------------+------------+
10 rows in set (0.00 sec)
```

Summary:

In this activity, we learnt,

1. Sqoop Import.
2. Data Processing using Hive.
3. Sqoop export.
4. Sqoop Connectors.
5. Hive Internal and External tables.