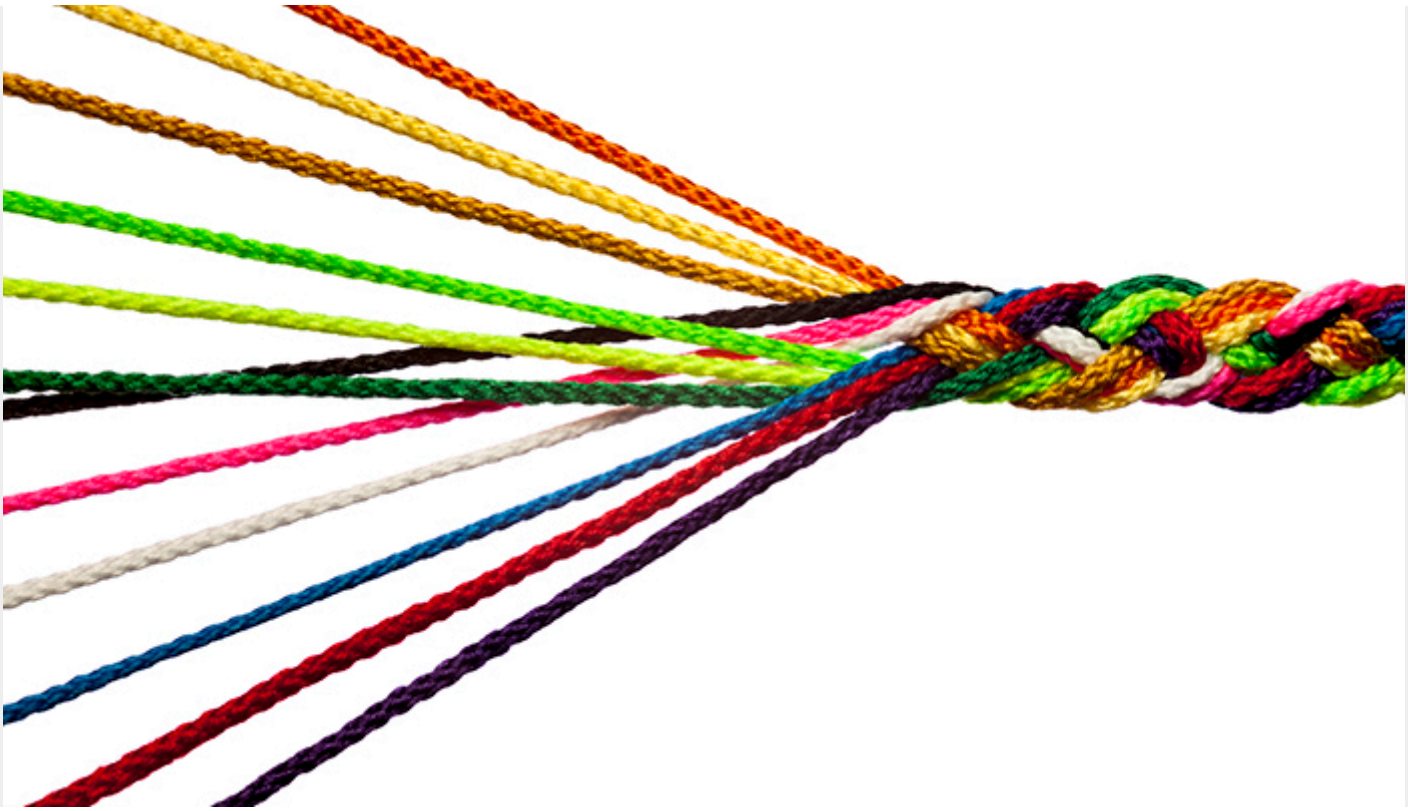
HOW TO WRITE A HIVE USER DEFINED FUNCTION (UDF) IN JAVA



Here we show how to write user defined functions (UDF) in Java and call that from Hive. You can then use a UDF in Hive SQL statements. It runs over whatever element you send it and then returns a result. So you would write a function to format strings or even do something far more complex.

In this example, we use this package:

com.example.hive.udf

This works with Hive primitives, meaning a string, integer, or other basic field types in Hive. It is very simple. You only have to write one function call, which implements the **evaluate()** interface. In the example below, we return a string, but you can return any primitive, such as:

```
public int evaluate();
public int evaluate(int a);
public double evaluate(int a, double b);
public String evaluate(String a, int b, Text c);
public Text evaluate(String a);
public String evaluate(List<Integer> a);
```

If you want to use complex Hive types, meaning **maps**, **arrays**, **structs**, or **unions**, you use:

org.apache.hadoop.hive.ql.udf.generic.GenericUDF,

There you have to implement two additional methods:

public ObjectInspector initialize(ObjectInspector[] arguments)
public String getDisplayString(String[] children)

And If you want to write user defined aggregation functions, you use:

org.apache.hadoop.hive.ql.exec.hive.UDAF

Aggregation functions are those that run over sets of data. For example, **average** runs over a set of data.

Technically, the code to do aggregate functions is not much more complicated to write. However, the architectural implementation is complex as this is a type of mapReduce function. That means it runs in parallel in a cluster, with pieces of the calculation coming together at different times. So you have to poll the **iterate()** function until Hive is done calculating the results then you can use that.

Sample Java Program

First, install Hive following these instructions.

Here we use **hive** as the CLI although **beehive** is a newer one that the Hive developers want users to switch too. Beehive includes all the same hive commands. It's only architecturally different. I prefer **hive**.

(**Note:** For the example below, it is easier to run everything as root. Otherwise you will get stuck on permissions issues and will spend time sorting those out. For example, the Hive process might not be able to read the Hadoop data because of file permissions. Adding further complexity, the userid you are using might be different than the user who is running Hive. You could, of course, work through those issues if you want.)

First, we need some data. We will use the Google daily year-to-date stock price. We will use this data in more than one blog post. Download it from here.

Shorten the long filename downloaded to **google.csv**. Delete the first line (headers) and the last two (includes summary info). Then you will have data like this:

"Feb 16, 2017","842.17","838.50","842.69","837.26","1.01M","0.58"
"Feb 15, 2017","837.32","838.81","841.77","836.22","1.36M","-0.32"
"Feb 14, 2017","840.03","839.77","842.00","835.83","1.36M","0.13"
"Feb 13, 2017","838.96","837.70","841.74","836.25","1.30M","0.49"
"Feb 10, 2017","834.85","832.95","837.15","830.51","1.42M","0.58"
"Feb 09, 2017","830.06","831.73","831.98","826.50","1.19M","0.02"
"Feb 08, 2017","829.88","830.53","834.25","825.11","1.30M","0.08"

As you can see, those quote marks are not helpful when working with numbers. So we will write a simple program to remove them.

First, this is Hive, so it will look to Hadoop for the data and not the local file system. So copy it to Hadoop.

hadoop fs -mkdir /data/stocks
hadoop fs -put /home/walker/Documents/bmc/google.csv /data/stocks

Here I have:

HADOOP_HOME=/usr/hadoop/hadoop-2.8.1
HIVE_HOME=/usr/local/hive/apache-hive-2.3.0-bin

We need jar files from there plus hadoop-common-2.8.1.jar, which you can download here.

So do that then:

export CLASSPATH=/usr/local/hive/apache-hive-2.3.0-bin/lib/hive-exec-2.3.0.jar:/usr/hadoop/hadoop-2.8.1/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.8.1.jar:/home/walker/Documents/bmc/hadoop-common-2.8.1.jar

Now, you do not need Maven or anything complicated like that. Just pick a folder where you want to write your Java file. Since we put the file in this package:

package com.example.hive.udf;

Java will expect that the .class files be in this folder structure:

com/example/hive/udf

So compile the code, copy the resulting .class file files to the package name folder, then create the JAR file:

javac Quotes.java
mv Quotes.class com/example/hive/udf/
jar cvf Quotes.jar com/example/hive/udf/Quotes.class

Code

Here is the code:

```
package com.example.hive.udf;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;
public final class Quotes extends UDF {
public Text evaluate(final Text s) {
if (s == null) { return null; }
return new Text(s.toString().replaceAll("^\"|\"$", ""));
}
}
```

Now, run hive and add the JAR file. (**Note** that you it from the local file system and not Hadoop.)

add jar /home/walker/Documents/bmc/Quotes.jar;

And to show that it is there:

list jars

Now create this table:

```
CREATE TABLE IF NOT EXISTS google(
tradeDate STRING,
price STRING,
open STRING,
```

```
high STRING,
low STRING,
vol STRING,
change STRING
)
COMMENT 'google stocks'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

And load data into it:

load data inpath '/home/walker/Documents/bmc/google.csv' into table google;

Now register it as a function. Note that that the class name is the package name plus the name of the Quotes class that we wrote:

create temporary function noQuotes as 'com.example.hive.udf.Quotes';

Then you can use the Hive UDF like this:

select noQuotes(price), price from google;

It will show:

2017 2017"
2017 2017"
2017 2017"
2017 2017"
2017 2017"
2017 2017"
2017 2017"
2017 2017"
2017 2017"

Note: The JAR file disappears when you exit Hive. So, for an exercise see how you can make Hive permanently store that.