Get stock data from google financial API, put this stream data continuously on kafka topic later read it from spark streaming program

We can fetch data from below URL

<http://finance.google.com/finance/info?client=ig&q>=

Here we need to append ticker information

Below is Java program to read from google web service and push it to Kafka topic

|  |
| --- |
| **package** org.cyb.stocks;  **import** java.io.BufferedReader;  **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.io.InputStreamReader;  **import** java.net.URL;  **import** java.net.URLConnection;  **import** java.nio.charset.Charset;  **import** java.util.Properties;  **import** org.apache.kafka.clients.producer.KafkaProducer;  **import** org.apache.kafka.clients.producer.Producer;  **import** org.apache.kafka.clients.producer.ProducerRecord;  **import** org.json.simple.JSONArray;  **import** org.json.simple.parser.JSONParser;  public **class** GoogleFinanceAPI {  public static void main(String[] args) {  String topicName = **null**;    String urlPrefix = "http://finance.google.com/finance/info?client=ig&q=";  Properties prop = **new** Properties();  InputStream input = **null**;  int intervalInSec = 0;    **try** {  input = **new** FileInputStream("config.properties");  prop.load(input);  } **catch** (IOException e) {  e.printStackTrace();  }  String tickers = prop.getProperty("tickers");  topicName = prop.getProperty("topic.name");  **try** {  intervalInSec = Integer.valueOf(prop.getProperty("interval"),30);  }**catch**(NumberFormatException ne){  intervalInSec =30;  }  //System.out.println(quote);  String url = urlPrefix+tickers;    Producer<String, String> producer = **new** KafkaProducer<String, String>(prop);  JSONParser parser = **new** JSONParser();  JSONArray jsonArr = **null**;    **try** {  **while**(**true**) {  String quotes = getQuotes(url);  quotes = quotes.substring(4);  //System.out.println("\nOutput: \n" + quotes);    jsonArr = (JSONArray) parser.parse(quotes);  String jsonQuotes = jsonArr.toJSONString();    **try**{  producer.send(**new** ProducerRecord<String, String>(topicName, jsonQuotes));  }**catch** (Exception e) {  e.printStackTrace();  }  System.out.println(jsonQuotes);  producer.flush();  Thread.sleep(intervalInSec\*1000);  }  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  producer.close ();  }    }  public static String getQuotes(String myURL) {  //System.out.println("Requeted URL:" + myURL);  StringBuilder sb = **new** StringBuilder();  URLConnection urlConn = **null**;  InputStreamReader in = **null**;  **try** {  URL url = **new** URL(myURL);  urlConn = url.openConnection();  **if** (urlConn != **null**)  urlConn.setReadTimeout(60 \* 1000);  **if** (urlConn != **null** && urlConn.getInputStream() != **null**) {  in = **new** InputStreamReader(urlConn.getInputStream(),  Charset.defaultCharset());  BufferedReader bufferedReader = **new** BufferedReader(in);  **if** (bufferedReader != **null**) {  int cp;  **while** ((cp = bufferedReader.read()) != -1) {  sb.append((char) cp);  }  bufferedReader.close();  }  }  in.close();  } **catch** (Exception e) {  **throw** **new** RuntimeException("Exception while calling URL:"+ myURL, e);  }    **return** sb.toString();  }  } |

Note: I haven’t implemented proper error handling

#config.properties file

|  |
| --- |
| #just put comma separated tickers  tickers = RELIANCE,MSFT,AAPL,GOOG  #tickers = NSE:RELIANCE,NASDAQ:MSFT  #interval in seconds  interval=5  #kafka properties  bootstrap.servers=localhost:9092,localhost:9093,localhost:9094  key.serializer=org.apache.kafka.common.serialization.StringSerializer  value.serializer=org.apache.kafka.common.serialization.StringSerializer  topic.name=mytopic  batch.size=16384 |

This program gets the quotes indefinitely every 5 seconds (configured in properties file) and pushes it to kafka topic

Note: we need to run zookeepers and kafka brokers before executing this program

Output will be JSON Array

|  |
| --- |
| [ { "id": "4674509" ,"t" : "RELIANCE" ,"e" : "NSE" ,"l" : "1,429.50" ,"l\_fix" : "1429.50" ,"l\_cur" : "₹1,429.50" ,"s": "0" ,"ltt":"11:33AM GMT+5:30" ,"lt" : "Jun 27, 11:33AM GMT+5:30" ,"lt\_dts" : "2017-06-27T11:33:32Z" ,"c" : "-6.35" ,"c\_fix" : "-6.35" ,"cp" : "-0.44" ,"cp\_fix" : "-0.44" ,"ccol" : "chr" ,"pcls\_fix" : "1435.85" } ,{ "id": "358464" ,"t" : "MSFT" ,"e" : "NASDAQ" ,"l" : "70.53" ,"l\_fix" : "70.53" ,"l\_cur" : "70.53" ,"s": "2" ,"ltt":"4:00PM EDT" ,"lt" : "Jun 26, 4:00PM EDT" ,"lt\_dts" : "2017-06-26T16:00:05Z" ,"c" : "-0.68" ,"c\_fix" : "-0.68" ,"cp" : "-0.95" ,"cp\_fix" : "-0.95" ,"ccol" : "chr" ,"pcls\_fix" : "71.21" ,"el": "70.53" ,"el\_fix": "70.53" ,"el\_cur": "70.53" ,"elt" : "Jun 26, 5:52PM EDT" ,"ec" : "0.00" ,"ec\_fix" : "0.00" ,"ecp" : "0.00" ,"ecp\_fix" : "0.00" ,"eccol" : "chb" ,"div" : "0.39" ,"yld" : "2.21" } ,{ "id": "22144" ,"t" : "AAPL" ,"e" : "NASDAQ" ,"l" : "145.82" ,"l\_fix" : "145.82" ,"l\_cur" : "145.82" ,"s": "2" ,"ltt":"4:00PM EDT" ,"lt" : "Jun 26, 4:00PM EDT" ,"lt\_dts" : "2017-06-26T16:00:05Z" ,"c" : "-0.46" ,"c\_fix" : "-0.46" ,"cp" : "-0.31" ,"cp\_fix" : "-0.31" ,"ccol" : "chr" ,"pcls\_fix" : "146.28" ,"el": "145.88" ,"el\_fix": "145.88" ,"el\_cur": "145.88" ,"elt" : "Jun 26, 7:59PM EDT" ,"ec" : "+0.06" ,"ec\_fix" : "0.06" ,"ecp" : "0.04" ,"ecp\_fix" : "0.04" ,"eccol" : "chg" ,"div" : "0.63" ,"yld" : "1.73" } ,{ "id": "304466804484872" ,"t" : "GOOG" ,"e" : "NASDAQ" ,"l" : "952.27" ,"l\_fix" : "952.27" ,"l\_cur" : "952.27" ,"s": "2" ,"ltt":"4:00PM EDT" ,"lt" : "Jun 26, 4:00PM EDT" ,"lt\_dts" : "2017-06-26T16:00:04Z" ,"c" : "-13.32" ,"c\_fix" : "-13.32" ,"cp" : "-1.38" ,"cp\_fix" : "-1.38" ,"ccol" : "chr" ,"pcls\_fix" : "965.59" ,"el": "953.73" ,"el\_fix": "953.73" ,"el\_cur": "953.73" ,"elt" : "Jun 26, 7:57PM EDT" ,"ec" : "+1.46" ,"ec\_fix" : "1.46" ,"ecp" : "0.15" ,"ecp\_fix" : "0.15" ,"eccol" : "chg" ,"div" : "" ,"yld" : "" } ] |

# First Approach

Spark program written in scala to read data from topic and write it to hdfs folder

|  |
| --- |
| **package** org.apache.spark  **import** java.util.HashMap  **import** org.apache.spark.streaming.\_  **import** org.apache.spark.streaming.kafka.\_  **import** org.apache.spark.sql.SQLContext  **import** org.apache.spark.sql.SaveMode  **object** StockConsumer {  **def** main(args: Array[*String*]): Unit = {  **if** (args.length < 4) {  System.err.println("Usage: StockConsumer <zkQuorum><group> <topics> <numThreads>")  System.exit(1)  }  **val** Array(zkQuorum, group, topics, numThreads) = args  **val** sparkConf = **new** SparkConf().setAppName("StockMarketConsumer").setMaster("local[2]")  // Set the Spark StreamingContext to create a DStream for every 2 seconds  **val** ssc = **new** StreamingContext(sparkConf, Seconds(2))    ssc.checkpoint("checkpoint")  // Map each topic to a thread  **val** topicMap = topics.split(",").map((\_, numThreads.toInt)).toMap  // Map value from the kafka message (k, v) pair  **val** quotes = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(\_.\_2)    **val** sqlContext = SQLContextSingleton.getInstance(ssc.sparkContext)  quotes.foreachRDD{rdd => {  **if**(!rdd.partitions.isEmpty) {  **import** sqlContext.implicits.\_  // Convert your data to a DataFrame, depends on the structure of your data  **val** df = sqlContext.read.json(rdd)  df.printSchema()  df.write.mode(SaveMode.Append).json("/user/shalaj/stocks")  }  }  }    //quotes.print()  //quotes.saveAsTextFiles("myoutput")  ssc.start()  ssc.awaitTermination()  }  **object** SQLContextSingleton {  @transient **private** **var** instance: SQLContext = \_    **def** getInstance(sparkContext: SparkContext): SQLContext = {  **if** (instance == **null**) {  instance = **new** SQLContext(sparkContext)  }  instance  }  }  } |

Now build the jar and submit spark job to hdfs cluster

|  |
| --- |
| spark-submit --class "org.apache.spark.StockConsumer" --master yarn --master yarn --executor-memory 512m --total-executor-cores 1 Spark16-0.0.1-SNAPSHOT.jar 172.27.147.51:2181,172.27.147.51:2182,172.27.147.51:2183 testgroup mytopic 1 |

now check /user/shalaj/stocks on hdfs

|  |
| --- |
| [shalaj@mac127 ~]$ hdfs dfs -ls /user/shalaj/stocks  Found 19 items  -rw-r--r-- 3 shalaj shalaj 0 2017-06-26 17:51 /user/shalaj/stocks/\_SUCCESS  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-012d4924-80b4-47b0-8be2-5de00eaf90bd  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-150b433e-0546-473b-b847-da481cd40e7d  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-2f3fb53d-0b44-4c1a-b01e-beb36871e186  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-36bae5d1-d0e0-40cc-950d-536c69eecd5e  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-416809a7-8d9a-463a-8fd0-2bd34afdc7bc  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-4b43bb47-218d-4f7b-b11e-0d7cda383db3  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-4e23e4ba-55ac-4910-8476-ef763634d64f  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-55461e84-bb70-4534-ba4f-07afc26229d4  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-5c4105ea-eff4-48cd-bbdd-f599b4f32630  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-60f2389d-a01e-4bb6-b9a3-422acb28358b  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-83c7a659-ab96-4e5f-a426-9096f92b51d5  -rw-r--r-- 3 shalaj shalaj 15920 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-a756fe14-842b-41fc-93ea-dd8dbf004124  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-a83b8006-27ea-43b6-a13c-f0416ffce5bd  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-af8185e7-ffb2-4572-aefa-5c35edcea96b  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:50 /user/shalaj/stocks/part-r-00000-b87a3869-324e-4bc4-aad4-08563ec4180a  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-c6e7722d-24c2-49da-9b30-a77ede201c60  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-fb2d4c2e-663a-4d04-ab66-5e2da002f088  -rw-r--r-- 3 shalaj shalaj 1589 2017-06-26 17:51 /user/shalaj/stocks/part-r-00000-fed9a304-e07c-431b-b1cb-4c3c03ae8b55 |

We will get part files containing json objects

|  |
| --- |
| { "id": "4674509" ,"t" : "RELIANCE" ,"e" : "NSE" ,"l" : "1,429.50" ,"l\_fix" : "1429.50" ,"l\_cur" : "₹1,429.50" ,"s": "0" ,"ltt":"11:33AM GMT+5:30" ,"lt" : "Jun 27, 11:33AM GMT+5:30" ,"lt\_dts" : "2017-06-27T11:33:32Z" ,"c" : "-6.35" ,"c\_fix" : "-6.35" ,"cp" : "-0.44" ,"cp\_fix" : "-0.44" ,"ccol" : "chr" ,"pcls\_fix" : "1435.85" }  ,{ "id": "358464" ,"t" : "MSFT" ,"e" : "NASDAQ" ,"l" : "70.53" ,"l\_fix" : "70.53" ,"l\_cur" : "70.53" ,"s": "2" ,"ltt":"4:00PM EDT" ,"lt" : "Jun 26, 4:00PM EDT" ,"lt\_dts" : "2017-06-26T16:00:05Z" ,"c" : "-0.68" ,"c\_fix" : "-0.68" ,"cp" : "-0.95" ,"cp\_fix" : "-0.95" ,"ccol" : "chr" ,"pcls\_fix" : "71.21" ,"el": "70.53" ,"el\_fix": "70.53" ,"el\_cur": "70.53" ,"elt" : "Jun 26, 5:52PM EDT" ,"ec" : "0.00" ,"ec\_fix" : "0.00" ,"ecp" : "0.00" ,"ecp\_fix" : "0.00" ,"eccol" : "chb" ,"div" : "0.39" ,"yld" : "2.21" }  { "id": "22144" ,"t" : "AAPL" ,"e" : "NASDAQ" ,"l" : "145.82" ,"l\_fix" : "145.82" ,"l\_cur" : "145.82" ,"s": "2" ,"ltt":"4:00PM EDT" ,"lt" : "Jun 26, 4:00PM EDT" ,"lt\_dts" : "2017-06-26T16:00:05Z" ,"c" : "-0.46" ,"c\_fix" : "-0.46" ,"cp" : "-0.31" ,"cp\_fix" : "-0.31" ,"ccol" : "chr" ,"pcls\_fix" : "146.28" ,"el": "145.88" ,"el\_fix": "145.88" ,"el\_cur": "145.88" ,"elt" : "Jun 26, 7:59PM EDT" ,"ec" : "+0.06" ,"ec\_fix" : "0.06" ,"ecp" : "0.04" ,"ecp\_fix" : "0.04" ,"eccol" : "chg" ,"div" : "0.63" ,"yld" : "1.73" }  ,{ "id": "304466804484872" ,"t" : "GOOG" ,"e" : "NASDAQ" ,"l" : "952.27" ,"l\_fix" : "952.27" ,"l\_cur" : "952.27" ,"s": "2" ,"ltt":"4:00PM EDT" ,"lt" : "Jun 26, 4:00PM EDT" ,"lt\_dts" : "2017-06-26T16:00:04Z" ,"c" : "-13.32" ,"c\_fix" : "-13.32" ,"cp" : "-1.38" ,"cp\_fix" : "-1.38" ,"ccol" : "chr" ,"pcls\_fix" : "965.59" ,"el": "953.73" ,"el\_fix": "953.73" ,"el\_cur": "953.73" ,"elt" : "Jun 26, 7:57PM EDT" ,"ec" : "+1.46" ,"ec\_fix" : "1.46" ,"ecp" : "0.15" ,"ecp\_fix" : "0.15" ,"eccol" : "chg" ,"div" : "" ,"yld" : "" } |

Now we can create hive External table and point it to that directory

|  |
| --- |
| CREATE EXTERNAL TABLE stockjsondata  (  json string  )  LOCATION '/user/shalaj/stocks/'; |

We can explode JSON and overwrite it to different table and get individual records

|  |
| --- |
| CREATE TABLE stockdata (  id string,  ticker string,  exchange\_name string,  last\_trade\_price double,  l\_fix double,  last\_trade\_with\_curr string,  last\_trade\_size bigint,  last\_trade\_time string,  last\_trade\_date\_time\_formatted string,  last\_trade\_date\_time string,  change string,  c\_fix string,  change\_percent string,  cp\_fix string,  ccol string,  prev\_close\_price double,  extended\_hour\_last\_trade\_price double,  el\_fix double,  el\_cur string,  elt string,  ec string,  ec\_fix string,  ecp string,  ecp\_fix string,  eccol string,  dividend double,  yld double  )  //------------------------------------------------------------------------------------------------------------------------------  INSERT OVERWRITE TABLE stockdata  select  a.id,  a.t,  a.e,  a.l,  a.l\_fix,  a.l\_cur,  a.s,  a.ltt,  a.lt,  a.lt\_dts,  a.c,  a.c\_fix,  a.cp,  a.cp\_fix,  a.ccol,  a.pcls\_fix,  a.el,  a.el\_fix,  a.el\_cur,  a.elt,  a.ec,  a.ec\_fix,  a.ecp,  a.ecp\_fix,  a.eccol,  a.dividend,  a.yld  from stockjsondata sjd  LATERAL VIEW json\_tuple (sjd.json,'id','t','e','l','l\_fix','l\_cur','s','ltt','lt','lt\_dts','c','c\_fix','cp','cp\_fix','ccol','pcls\_fix','el','el\_fix','el\_cur','elt','ec','ec\_fix','ecp','ecp\_fix','eccol','div','yld')a  as  id,t,e,l,l\_fix,l\_cur,s,ltt,lt,lt\_dts,c,c\_fix,cp,cp\_fix,ccol,pcls\_fix,el,el\_fix,el\_cur,elt,ec,ec\_fix,ecp,ecp\_fix,eccol,dividend,yld; |

Or we can use jsonserde to directly parse the data from output folder



|  |
| --- |
| add jar /home/shalaj/json-serde-1.3.6-jar-with-dependencies.jar; |

We can add this jar to hive session and create external table which will point to data location

|  |
| --- |
| CREATE EXTERNAL TABLE external\_json (  id string,  t string,  e string,  l string,  l\_fix string,  l\_cur string,  s string,  ltt string,  lt string,  lt\_dts string,  c string,  c\_fix string,  cp string,  cp\_fix string,  ccol string,  pcls\_fix string,  el string,  el\_fix string,  el\_cur string,  elt string,  ec string,  ec\_fix string,  ecp string,  ecp\_fix string,  eccol string,  `div` string,  yld string  )  ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  LOCATION '/user/shalaj/stocks/'; |

# Second Approach

Iinstead of loading data in single folder we can load data into separate date folder like

**/home/shalaj/strocks/date=2017-06-27**

And later we can create one external partition table which can read data as per partition

Updated spark program

|  |
| --- |
| **package** org.apache.spark  **import** java.util.HashMap  **import** org.apache.spark.streaming.\_  **import** org.apache.spark.streaming.kafka.\_  **import** org.apache.spark.sql.SQLContext  **import** org.apache.spark.sql.SaveMode  **import** java.util.Calendar  **import** java.text.SimpleDateFormat  **object** StockConsumer {  **def** main(args: Array[*String*]): Unit = {  **if** (args.length < 4) {  System.err.println("Usage: StockConsumer <zkQuorum><group> <topics> <numThreads>")  System.exit(1)  }  **val** Array(zkQuorum, group, topics, numThreads) = args  **val** sparkConf = **new** SparkConf().setAppName("StockMarketConsumer").setMaster("local[2]")  // Set the Spark StreamingContext to create a DStream for every 2 seconds  **val** ssc = **new** StreamingContext(sparkConf, Seconds(2))    ssc.checkpoint("checkpoint")  // Map each topic to a thread  **val** topicMap = topics.split(",").map((\_, numThreads.toInt)).toMap  // Map value from the kafka message (k, v) pair  **val** quotes = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(\_.\_2)    **val** sqlContext = SQLContextSingleton.getInstance(ssc.sparkContext)  **val** now = Calendar.getInstance().getTime()  **val** format = **new** SimpleDateFormat("YYYY-MM-dd")  **val** currentDay = format.format(now)  quotes.foreachRDD{rdd => {  **if**(!rdd.partitions.isEmpty) {  **import** sqlContext.implicits.\_  // Convert your data to a DataFrame, depends on the structure of your data  **val** df = sqlContext.read.json(rdd)  df.printSchema()  df.write.mode(SaveMode.Append).json("/user/shalaj/stocks/date="+currentDay)  }  }  }    //quotes.print()  //quotes.saveAsTextFiles("myoutput")  ssc.start()  ssc.awaitTermination()  }  **object** SQLContextSingleton {  @transient **private** **var** instance: SQLContext = \_    **def** getInstance(sparkContext: SparkContext): SQLContext = {  **if** (instance == **null**) {  instance = **new** SQLContext(sparkContext)  }  instance  }  }  } |

After running program check hdfs directory path



Now we can see folder is created as per date

Now we can create partition table

|  |
| --- |
| CREATE EXTERNAL TABLE stockpartitionjsondata  (  json string  )  PARTITIONED BY (date string)  LOCATION '/user/shalaj/stocks/'; |

Run select query on table

|  |
| --- |
|  |

You will not get any data since you have not added partition in metadata so we need to manually create partition that will be added to metadata than only we can get the results from this table

Run below command

|  |
| --- |
| MSCK REPAIR TABLE stockpartitionjsondata; |

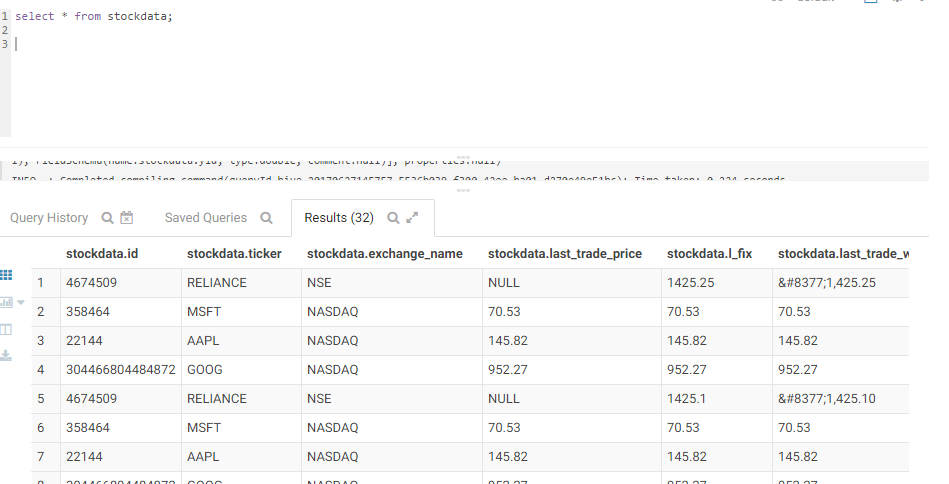
|  |
| --- |
|  |

Now again execute select query

|  |
| --- |
|  |

Now we can use insert into command to store the data into managed table

|  |
| --- |
| INSERT INTO TABLE stockdata  select  a.id,  a.t,  a.e,  a.l,  a.l\_fix,  a.l\_cur,  a.s,  a.ltt,  a.lt,  a.lt\_dts,  a.c,  a.c\_fix,  a.cp,  a.cp\_fix,  a.ccol,  a.pcls\_fix,  a.el,  a.el\_fix,  a.el\_cur,  a.elt,  a.ec,  a.ec\_fix,  a.ecp,  a.ecp\_fix,  a.eccol,  a.dividend,  a.yld  from stockpartitionjsondata sjd  LATERAL VIEW json\_tuple (sjd.json,'id','t','e','l','l\_fix','l\_cur','s','ltt','lt','lt\_dts','c','c\_fix','cp','cp\_fix','ccol','pcls\_fix','el','el\_fix','el\_cur','elt','ec','ec\_fix','ecp','ecp\_fix','eccol','div','yld')a  as  id,t,e,l,l\_fix,l\_cur,s,ltt,lt,lt\_dts,c,c\_fix,cp,cp\_fix,ccol,pcls\_fix,el,el\_fix,el\_cur,elt,ec,ec\_fix,ecp,ecp\_fix,eccol,dividend,yld  where date='2017-06-27'; |



By this approach we can insert the data on daily basis by using partitioned column