# **Data Ingest:**

## **Import data from a MySQL database into HDFS using Sqoop :**

### List the tables in the Loudacre database :

|  |
| --- |
| $ *sqoop list-tables --connect jdbc:mysql://localhost/loudacre --username training --password training* |

### Use Sqoop to import the accounts table in the loudacre database and save it in HDFS under /loudacre

|  |
| --- |
| *$ sqoop import --connect jdbc:mysql://localhost/loudacre --username training --password training --table accounts --target-dir /loudacre/accounts --null-non-string ' \\N’* |

### Importing Incremental Updates :

|  |
| --- |
| *$ sqoop import \*  *--connect jdbc:mysql://localhost/loudacre \*  *--username training --password training \*  *--incremental append \*  *--null-non-string ' \\N' \*  *--table accounts \*  *--target-dir /loudacre/accounts \*  *--check-column acct\_num \*  *--last-value <largest\_acct\_num>* |

### Evaluate the data

|  |
| --- |
| *$ sqoop eval --query "SELECT \* FROM webpage LIMIT 10" \*  *--connect jdbc:mysql://localhost/loudacre \*  *--username training --password training* |

### Sqoop’s incremental lastmodified mode imports new and modified records :

|  |
| --- |
| *$ sqoop import --table invoices --connect jdbc:mysql://dbhost/loudacre \*  *--username dbuser --password pw \*  *--incremental lastmodified \*  *--check-column mod\_dt \*  *--last-value ' 2015-09-30 16:00:00'* |

### Use Sqoop’s incremental append mode to import only new records Based on value of last record in specified column :

|  |
| --- |
| *$ sqoop import --table invoices \*  *--connect jdbc:mysql://dbhost/loudacre \*  *--username dbuser --password pw \*  *--incremental append \*  *--check-column id \*  *--last-value 9478306* |

### Import only specific columns from account table :

|  |
| --- |
| *$ sqoop import --table accounts \*  *--connect jdbc:mysql://dbhost/loudacre \*  *--username dbuser --password pw \*  *--columns "id,first\_name,last\_name,state"* |

### Import only matching rows from accounts table

|  |
| --- |
| *$ sqoop import --table accounts \*  *--connect jdbc:mysql://dbhost/loudacre \*  *--username dbuser --password pw \*  *--where "state='CA'"* |

### Using a Free-Form Query :

|  |
| --- |
| *$ sqoop import \ --connect jdbc:mysql://dbhost/loudacre \ --username dbuser --password pw \ --target-dir /data/loudacre/payable \ --split-by accounts.id \ --query 'SELECT accounts.id, first\_name, last\_name, bill\_amount FROM accounts JOIN invoices ON (accounts.id = invoices.cust\_id) WHERE $CONDITIONS'* |

### Using a Free-Form Query with WHERE Criteria :

|  |
| --- |
| *$ sqoop import \ --connect jdbc:mysql://dbhost/loudacre \ --username dbuser --password pw \ --target-dir /data/loudacre/payable \ --split-by accounts.id \ --query 'SELECT accounts.id, first\_name, last\_name, bill\_amount FROM accounts JOIN invoices ON (accounts.id = invoices.cust\_id) WHERE $CONDITIONS AND bill\_amount >= 40'* |

### Controlling Parallelism : you can increase the number of tasks :

|  |
| --- |
| *$ sqoop import --table accounts \ --connect jdbc:mysql://dbhost/loudacre \ --username dbuser --password pw \ -m 8* |

## Export data to a MySQL database from HDFS using Sqoop:

### Exporting Data from Hadoop to RDBMS with Sqoop :

|  |
| --- |
| *$ sqoop export \*  *--connect jdbc:mysql://dbhost/loudacre \*  *--username dbuser --password pw \*  *--export-dir /loudacre/recommender\_output \*  *--update-mode allowinsert \*  *--table product\_recommendations* |

## Change the delimiter and file format of data during import using Sqoop :

### import the webpage table, but use the tab character (\t) instead of the default (comma) as the field terminator:

|  |
| --- |
| *$ sqoop import --connect jdbc:mysql://localhost/loudacre \*  *--username training --password training \*  *--table webpage \*  *--target-dir /loudacre/webpage \*  *--fields-terminated-by "\t“* |

### Ingest real-time and near-real time (NRT) streaming data into HDFS using Flume

#### Configuring Flume Components :

##### Spool directory :

|  |
| --- |
| *agent1.sources = src1*  *agent1.sinks = sink1*  *agent1.channels = ch1*  *agent1.channels. ch1.type = memory*  *agent1.sources. src1.type = spooldir*  *agent1.sources. src1.spoolDir = /var/flume/incoming*  *agent1.sources. src1.channels = ch1*  *agent1.sinks. sink1.type = hdfs*  *agent1.sinks. sink1.hdfs.path = /loudacre/logdata*  *agent1.sinks. sink1.channel = ch1* |

##### netcat

|  |
| --- |
| *myagent.sources= mysrc1*  *myagent.channels = mych1*  *myagent.sinks = mysink1*  *myagent.channels.mych1.type = memory*  *myagent.sources.mysrc1.type = netcat*  *myagent.sources.mysrc1.bind = localhost*  *myagent.sources.mysrc1.port = 12345*  *myagent.sources.mysrc1.channels = mych1*  *myagent.sinks.mysink1.channel = mych1*  *myagent.sinks.mysink1.type = logger*  *myagent.channel.mych1.capacity=10000*  *myagent.channel.mych1.transactionCapacity=10000* |

#### HDFS Sink Configuration :

##### **Specifying Pattern And Commpression Type:**

|  |
| --- |
| *agent1.sinks.sink1.type = hdfs*  *agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d*  *agent1.sinks.sink1.hdfs.codeC = snappy*  *agent1.sinks.sink1.channel = ch1* |

##### **Setting fileType parameter to DataStream writes raw data**

|  |
| --- |
| *agent1.sinks.sink1.type = hdfs*  *agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d*  *agent1.sinks.sink1.hdfs.fileType = DataStream*  *agent1.sinks.sink1.hdfs.fileSuffix = .txt*  *agent1.sinks.sink1.channel = ch1* |

##### **Strating Fulme Agent**

|  |
| --- |
| *$ flume-ng agent \*  *--conf /etc/flume-ng/conf \*  *--conf-file /path/to/flume.conf \*  *--name agent1 \*  *-Dflume.root.logger=INFO,console* |

# Load data into and out of HDFS using the Hadoop File System (FS) commands

Directory Listing :

|  |
| --- |
| *$ hdfs dfs –ls*  *$ hdfs dfs –ls /* |

Copy file:

|  |
| --- |
| *$ hdfs dfs –put foo.txt foo\_hfds.txt* |

Display the contents of the HDFS file

|  |
| --- |
| *$ hdfs dfs –cat /Loudacre/fred/bar.txt* |

### Copy that file to the local disk

|  |
| --- |
| *$ hdfs dfs –get /user/fred/bar.txt barlocal.txt* |

### Creating Directory in HDFS

|  |
| --- |
| *$ hdfs dfs –mkdir myhdfsdirectory* |

### Delete Directory from HDFS

|  |
| --- |
| *$ hdfs dfs –rm –r myhdfsdirectory* |

# **Transform, Stage, Store**

## Load data from HDFS and store results back to HDFS using Spark

#### Load/Save Text File

|  |
| --- |
| ***Pyhton:***  *rdd=sc.textFile("/loudacre/weblogs/FlumeData.1463945071536")*  *rdd.saveAsTextFile("/loudacre/spark.txt")*  ***Scala:***  *Ss* |

#### Load/ Save Sequence File

#### Load/ Save Avro File

|  |
| --- |
| ***Pyhton:***  *# Creates a DataFrame from a directory*  *df = sqlContext.read.format("com.databricks.spark.avro").load("input dir")*  *# Saves the subset of the Avro records read in*  *df.where("age > 5").write.format("com.databricks.spark.avro").save("output dir")*  ***Scala:***  *import com.databricks.spark.avro.\_*  *val sqlContext = new SQLContext(sc)*  *val df = sqlContext.read.format("com.databricks.spark.avro").load("input dir")*  *df.filter("age > 5").write.format("com.databricks.spark.avro").save("output dir")* |

#### Load JSON file

|  |
| --- |
| *sqlContext.jsonFile('python/test\_support/sql/people.json').dtypes* |

#### Loading Parquet File

|  |
| --- |
| *sqlContext.parquetFile('python/test\_support/sql/parquet\_partitioned').dtypes* |

### Join disparate datasets together using Spark :

|  |
| --- |
| *# Step 1 - Create an RDD based on a subset of weblogs (those ending in digit 6)*  *logs=sc.textFile("/loudacre/weblogs/\*6")*  *# map each request (line) to a pair (userid, 1), then sum the values*  *userreqs = logs.map(lambda line: line.split()) .map(lambda words: (words[2],1)) \*  *.reduceByKey(lambda count1,count2: count1 + count2)*    *# Step 2 - Show the records for the 10 users with the highest counts*  *freqcount = userreqs.map(lambda (userid,freq): (freq,userid)).countByKey()*  *print freqcount*  *# Step 3 - Group IPs by user ID*  *userips = logs .map(lambda line: line.split()) .map(lambda words: (words[2],words[0])) .groupByKey()*  *# print out the first 10 user ids, and their IP list*  *for (userid,ips) in userips.take(10):*  *print userid, ":"*  *for ip in ips: print "\t",ip*  *# Step 4a - Map account data to (userid,[values....])*  *accounts = sc.textFile("/loudacre/accounts").map(lambda s: s.split(',')) \*  *.map(lambda account: (account[0],account))*  *# Step 4b - Join account data with userreqs then merge hit count into valuelist*  *accounthits = accounts.join(userreqs)*  *# Step 4c - Display userid, hit count, first name, last name for the first 5 elements*  *for (userid,(values,count)) in accounthits.take(5) :*  *print userid, count, values[3],values[4]*    *# Challenge 1 - key accounts by postal/zip code*  *accountsByPCode = sc.textFile("/loudacre/accounts") .map(lambda s: s.split(',')).keyBy(lambda account: account[8])*    *# Challenge 2 - map account data to lastname,firstname*  *namesByPCode = accountsByPCode\*  *.mapValues(lambda account: account[4] + ',' + account[3]) \*  *.groupByKey()*  *# Challenge 3 - print the first 5 zip codes and list the names*  *for (pcode,names) in namesByPCode.sortByKey().take(5):*  *print "---" ,pcode*  *for name in names: print name* |

#### Join 2 RDD

#### LeftOuter Join

### Calculate aggregate statistics (e.g., average or sum) using Spark

### Filter data into a smaller dataset using Spark

### Write a query that produces ranked or sorted data using Spark

# **Data Analysis**

## Read and/or create a table in the Hive metastore in a given schema

## Extract an Avro schema from a set of datafiles using avro-tools

### Use the avro-tools command to work with binary files

|  |
| --- |
| *$ avro-tools tojson mydatafile.avro*  *$ avro-tools getschema mydatafile.avro* |

## Create a table in the Hive metastore using the Avro file format and an external schema file

### Using Sqoop :

Sqoop supports importing data as Avro, or exporting data from existing Avro data files ,sqoop importsaves the schema JSON file in local directory :

|  |
| --- |
| *$ sqoop import \*  *--connect jdbc:mysql://localhost/loudacre \*  *--username training --password training \*  *--table accounts \*  *--target-dir /loudacre/accounts\_avro \*  *--as-avrodatafile*  *--hive table accounts\_avro* |

### Hive Impala Table creation from schema in a separate file :

|  |
| --- |
| *CREATE TABLE order\_details\_avro*  *STORED AS AVRO*  *TBLPROPERTIES (' avro.schema.url' = 'hdfs://localhost/loudacre/accounts\_schema.json');* |

### Hive Impala Table creation from schema in-line:

|  |
| --- |
| *CREATE TABLE order\_details\_avro*  *STORED AS AVRO*  *TBLPROPERTIES (' avro.schema.literal' =*  *'{"name": "order",*  *"type": "record",*  *"fields": [*  *{"name":"order\_id", "type":"int"},*  *{"name":"cust\_id", "type":"int"},*  *{"name":"order\_date", "type":"string"} ]}');* |

## Improve query performance by creating partitioned tables in the Hive metastore

### Impala & Hive Partitioning

|  |
| --- |
| *CREATE EXTERNAL TABLE accounts\_by\_state(*  *cust\_id INT, fname STRING,lname STRING,address STRING,city STRING,*  *zipcode STRING)*  *PARTITIONED BY (state STRING)*  *ROW FORMAT DELIMITED*  *FIELDS TERMINATED BY ','*  *LOCATION '/loudacre/accounts\_by\_state';* |

### Nested Partitions:

|  |
| --- |
| *CREATE EXTERNAL TABLE accounts\_by\_state(*  *cust\_id INT, fname STRING,lname STRING,address STRING,city STRING,*  *zipcode STRING)*  *PARTITIONED BY (state STRING, zipcode STRING)*  *ROW FORMAT DELIMITED*  *FIELDS TERMINATED BY ','*  *LOCATION '/loudacre/accounts\_by\_state';* |

### Create new partition Dynamically from existing data :

|  |
| --- |
| *INSERT OVERWRITE TABLE accounts\_by\_state*  *PARTITION(state)*  *SELECT cust\_id, fname, lname, address,*  *city, zipcode, state FROM accounts;* |

### Static partitioing is the same as dynamic partitioing in term of command

|  |
| --- |
| *CREATE TABLE call\_logs (*  *call\_time STRING, phone STRING, event\_type STRING,details STRING)*  *PARTITIONED BY (call\_date STRING)*  *ROW FORMAT DELIMITED*  *FIELDS TERMINATED BY ',';*  *-----------------------*  *ALTER TABLE call\_logs*  *ADD PARTITION (call\_date=' 2014-10-02' ) ;* |

### Load Data into static partition :

|  |
| --- |
| *LOAD DATA INPATH ' /mystaging/call-20141002.log'*  *INTO TABLE call\_logs*  *PARTITION(call\_date=' 2014-10-02') ;* |

### Overwrite Data in Partition

|  |
| --- |
| *LOAD DATA INPATH ' /mystaging/call-20141002.log'*  *INTO TABLE call\_logs OVERWRITE*  *PARTITION(call\_date=' 2014-10-02') ;* |

### View current partitions in a Table:

|  |
| --- |
| *SHOW PARTITIONS call\_logs;* |

### Alter Table and add partition :

|  |
| --- |
| *ALTER TABLE call\_logs*  *ADD PARTITION (call\_date='2013-06-05')*  *LOCATION '/loudacre/call\_logs/call\_date=2013-06-05';* |

### Drop Partition from Table :

|  |
| --- |
| *ALTER TABLE call\_logs*  *DROP PARTITION (call\_date='2013-06-06');* |

### MSCK REPAIR

|  |
| --- |
| *MSCK REPAIR TABLE call\_logs;* |

### Enabling dynamic partitioning in Hive Older Version :

|  |
| --- |
| *SET hive.exec.dynamic.partition=true;*  *SET hive.exec.dynamic.partition.mode=nonstrict;* |

### Hive configuration to limit create manay partitions

* Maximum number if dynamic partitions that can be create by any given node involved in a query ,default is 100

|  |
| --- |
| ***hive.exec.max.dynamic.partitions.pernode*** |

* Total number of dynamic partitions that can be create by one HiveQL statment, defult is 1000:

|  |
| --- |
| ***hive.exec.max.dynamic.partitions*** |

* Maximum total files (on all nodes) created by a query , Default 100000:

|  |
| --- |
| ***hive.exec.max.dynamic.partitions*** |

## Evolve an Avro schema by changing JSON files