ISIT312/ISIT912 Big Data Management

Spring 2023

Using Apache Pig Latin

In this practice, you will learn how to use Apache Pig Latin for processing of the data files stored in HDFS.

DO NOT attempt to copy the Linux commands in this document to your working Terminal, because it is error-prone. Type those commands by yourself.

Laboratory Instructions.

(0) Start Hadoop
Start five Hadoop services (see instructions in a previous lab).

(1) Pig Grunt shell
You can use Pig Grunt command-line shell to interact with Pig. To use the later, in a Terminal window, enter the following command:

```
$PIG_HOME/bin/pig
```

You will see a command line with the `grunt>` prompt.

(2) Upload data to HDFS
Use Terminal to upload data to HDFS. To do so, first create a text file `orders.txt` with the following contents:

```
bolt,James,200,2016,01,01
bolt,Peter,100,2017,01,30
bolt,Bob,300,2018,05,23
screw,James,20,2017,05,11
screw,Alice,55,2018,01,01
nut,Alice,23,2018,03,16
washer,James,45,2016,04,24
washer,Peter,100,2016,05,12
bolt,James,200,2018,01,05
bolt,Peter,100,2018,01,05
bolt,James,,2018,01,01
```

Use Terminal to upload the file into HDFS into a location `/user/bigdata` in the following way:

```
$HADOOP_HOME/bin/hadoop fs -put orders.txt /user/bigdata
```

(3) How to load and dump data in Pig

To load sample data into a Pig data container called `orders` and retrieve the data, process the following Pig's command in front of `grunt>` prompt:

```
orders = load '/user/bigdata/orders.txt' using PigStorage(',');
```

To list the contents of a data container `orders`, process the following Pig's command in front of `grunt>` prompt:

```
dump orders;
```

Note, that a clause `PigStorage(',')` determines a separator between the values in the rows of input data file. By default a separator is `TAB`. In our case the values are separated with a comma.

(4) How to assign the names and types to columns in Pig data container ?
To assign the names and types to columns in Pig data container process the following command:

```
orders = load '/user/bigdata/orders.txt' using PigStorage(',') as
(item:chararray,customer:chararray,quantity:int,year:int,month:int,day:int);
```

A clause `as ( ... )` determines the names of columns and the types of columns.

To verify the results process a command `describe`.

```
describe orders;
```

(5) How to load a map into Pig storage ?
Create a text file `keyvalue.txt` that contains the following lines:

```
[first-name#James,last-name#Bond,city#London,country#UK]
[first-name#Harry,last-name#Potter,city#Avondale,country#UK]
```

Next, use Terminal to process the following command that loads a file `keyvalue.txt` into HDFS folder `/user/bigdata`:

```
$HADOOP_HOME/bin/hadoop fs -put keyvalue.txt /user/bigdata
```

Then, to create a new data container `keyvalue`, process the following command in front of `grunt>` prompt:

```
keyvalue = load '/user/bigdata/keyvalue.txt' as (personal:map[]);
```

Next, at `grunt>` prompt, process `describe` and `dump` statements to list the structures and contents of a data container `keyvalue`.

```
describe keyvalue;
dump keyvalue;
```

Next, at `grunt>` prompt, process `foreach` statement to create a new data container `keyvalue-city` that contains only the names of cities.

```
keyvalue_city = foreach keyvalue generate personal#'city';
```

Next, at grunt> prompt, process describe and dump statements to list the structures and contents of a data container keyvalue_city.

```
describe keyvalue_city;
dump keyvalue_city;
```

A data container keyvalue contains two maps earlier loaded into HDFS. A new data container keyvalue_city contains only the values associated with a key city.

(6) How to load a bag with tuples into Pig storage ?

Create a text file hobbies.txt with the following rows:

```
James,({painting},{swimming})
Harry,({cooking})
Robin,({})
```

The file contains information about the first names of people and their hobbies.

Next, use Terminal to process the following command that loads a file hobbies.txt into HDFS folder /user/bigdata:

```
$HADOOP_HOME/bin/hadoop fs -put hobbies.txt /user/bigdata
```

Then, to create a new data container called hobbies, process the following command in front of grunt> prompt:

```
hobbies = load '/user/bigdata/hobbies.txt' as
(firstname:chararray,hobbies:bag{t:(hobby:chararray)});
```

Next, at grunt> prompt, process describe and dump statements to list the structures and contents of a data container hobbies.

```
dump hobbies;
describe hobbies;
```

Note, that a name of a bag t must be included in a specification of bag structure. It is also possible to nest within a bag not only sets of values but also sets of tuples.

Create a new text file nested.txt with the following contents:

```
James,({Ferrari,xyz123}{Honda,pkr856})
Harry,({Rolls Royce,xxx666})
```

Next, use Terminal to process the following command that loads a file nested.txt into HDFS folder /user/bigdata:

```
$HADOOP_HOME/bin/hadoop fs -put nested.txt /user/bigdata
```

Then, to create a new data container nested, process the following command in front of grunt> prompt:

```
nested = load '/user/bigdata/nested.txt' as (
    firstname:chararray,
    cars:bag{t:(
              manufacturer:chararray,
              rego:chararray)}
    );
```

Next, at `grunt>` prompt, process `describe` and `dump` commands to list the structures and contents of a data container `nested`.

```
dump nested;
describe nested;
```

(7) How to compute projections of a data container ?

Assume that we would like to create a container `items` that contains only the names of items extracted from a container `orders`. First we create a container `orders` using a file `orders.txt` with load command:

```
orders = load '/user/bigdata/orders.txt' using PigStorage(',')
as (item:chararray,customer:chararray,
quantity:int,year:int,month:int,day:int);
```

Next, we use `foreach` command to create a new container `items` as projection of `orders` on a column `item` and finally we list the contents of a container `items` with `dump` command:

```
items = foreach orders generate item;
dump items;
```

To remove the duplicates we apply `distinct` command to a container `items` and next we use the commands `dump` and `describe` to list the contents and structure of a new container `distinctitems`.

```
distinctitems = distinct items;
dump distinctitems;
describe distinctitems;
```

In the same command `distinct` can be used to eliminated the duplicate from the pairs, triples, … , n-tuples of columns.

```
dates = foreach orders generate day, month, year;
dump dates;
```

(8) How to perform selections from data container ?

A `filter` command can be used to filter the contents of a data container. For example, we can select all orders where *a quantity is greater than 100* in the following way:

```
biggerorders = filter orders by quantity > 100;
dump biggerorders;
```

It is always an interesting problem how to find that rows that have no value in a column. It means to filter the rows with null condition.

```
nulls = filter orders by quantity is null;
dump nulls;
```

(9) How to split data containers ?

Sometimes, it is convenient to save the rows filtered from a data container into several other containers. For example, we split a data container `orders` into a container `orders2018` that contains orders submitted in `2018` and a data container `olderorders` that contains other orders. Such split of a data container `orders` can be performed with `split` command in the following way:

```
split orders into
  orders2018 if year==2018,
  olderorders otherwise;
dump orders2018;
```

(10) How to compute an inner join ?

Create a new data set `items.txt` with the following contents:

```
bolt,2.23
screw,3.5
nut,1.25
washer,2.5
nail,0.4
fastener,4.1
pin,10.05
coupler,9.95
```

Upload the new data set into HDFS into a folder `/user/bigdata` in the following way:

```
$HADOOP_HOME/bin/hadoop fs -put items.txt /user/bigdata
```

Next, create a new data container `items` and list the contents of the container in the following way:

```
items = load '/user/bigdata/items.txt' using PigStorage(',') as
             (item:chararray,price:float);
dump items;
```

We would like to find the prices of all ordered product. To do so we have to join the data containers `orders` and `items` over a column `item` in both containers. The structures of the data containers orders and items are the following:

```
items: {item: chararray,
        price: float}

orders: {item: chararray,
         customer: chararray,
         quantity: int,
         year: int,
         month: int,
         day: int}
```

A new data container `inner_join` is created by joining the data containers `orders` and `items` in the following way:

```
inner_join = join orders by item, items by item;
```

To display the structures and the contents of a new data container `inner_join` process the following commands:

```
describe inner_join;
dump inner_join;
```

(11) How to implement left outer join ?

A left outer join operation joins all rows in a data container `items` with all rows in a data container `orders` and additionally includes into the results the rows from a data container `items` that cannot be joined with any rows from a data container `orders`. Such rows from `items` are extended with `NULL`s in all columns coming from a data container `orders`. Such rows represent the items that have never been included in any order so far. Process the following command:

```
leftouter_join = join items by item left outer, orders by item;
```

To display the structures and the contents of a new data container `leftouter_join` process the following commands:

```
dump leftouter_join;
describe leftouter_join;
```

The results saved in a data container `leftouter_join` can be used to find the names of items that have not been ordered yet. It is nothing else but implementation of *antijoin* operation. *Antijoin* operation, in contrast to join operation, finds all rows from the left argument of the operation that cannot be joined with the rows from the right argument of the operation. Process the following commands to find all items that have not been ordered yet.

```
notordered = filter leftouter_join by orders::item is null;
```

To display information about the items that have never been ordered yet process the following command:

```
dump notordered;
```

(12) How to implement non-equi join ?

An inner join operation computed in a step 10 assumes that the rows are connect only when a value in a column `item` in a container `items` is the same as a value in a column `item` in a container `orders`. It is so called equi join. It is possible to implement a join operation that joins the rows from two containers that satisfy any condition different from equality condition. For example, find all pairs of items such that the first element in each pair has a price higher than the second element. Process the following sequence of commands and verify the results after each step:

```
newitems = load '/user/bigdata/items.txt' using PigStorage(',')
```

```
            as (item:chararray,price:float);

  crossjoin = cross items, newitems;

  describe crossjoin;

  result = filter crossjoin by items::price > newitems::price;

  dump result;
```

(13) How to perform groupings and how to compute aggregation functions ?

An operation `group` can be used to restructure a container with the flat tuples into a container with the tuples nested within bags.

Assume that we would like to aggregate all orders on the same items into the bags and assign these bags with the ordered item. This can be achieved in the following way:

```
  ordergrp = group orders by item;
```

To display the structures of a data container `ordergrp` obtained after grouping, process the following command:

```
  describe ordergrp;
```

The structures of a data container `ordergrp` are the following:

```
ordergrp: {group: chararray,
           orders: {(item: chararray,
                     customer: chararray,
                     quantity: int,
                     year: int,
                     month: int,
                     day: int)}}
```

To display the contents of a data container `ordergrp` obtain after grouping, process the following command:

```
  dump ordergrp;
```

Now, it is possible to count the total number of orders in each bag to get the result identical to `SELECT` with `GROUP BY` clause of SQL. Process the following `foreach` command:

```
  itemscnt = foreach ordergrp generate group, COUNT(orders.item);
```

Verify the contents of a data container `itemscnt` in the following way:

```
  dump itemscnt;
```

(14) How to process `CUBE` and `ROLLUP` operators ?

`CUBE` operator performs grouping over all subsets of a given set of columns and saves the results in a bag.

Process the following commands:

```
ordcube = cube orders by CUBE(item, customer);
describe ordcube;
dump ordcube;
```

Like with `group` operation, we can apply an aggregation function to the results.

```
cntcube = foreach ordcube generate group, COUNT(cube.item);
dump cntcube;
```

In the same way we can process `ROLLUP` operator:

```
ordrollup = cube orders by ROLLUP(item,customer);
dump ordrollup;
describe ordrollup;
```

In the same way as before, we can apply an aggregation function to the results.

```
cntrollup = foreach ordrollup generate group, COUNT(cube.item);
dump cntrollup;
```

(15) How to unnest (flatten) data bags ?

Consider a data container `ordergrp` created in a step (13) an list its structures in the following way:

```
describe ordergrp;
```

The structures of a data container `ordergrp` are the following:

```
ordergrp: {group: chararray,orders: {(item: chararray,customer:
chararray,quantity: int,year: int,month: int,day: int)}};
```

An operation `flatten` can be used to "ungroup" nested structures. Process the following commands:

```
orderunnest = foreach ordergrp generate flatten (orders);
describe orderunnest;
dump orderunnest;
```

(16) How to restrict outputs ?

To restrict the outpust to a given number of records, we can use `limit` operation. The following command gets the first 4 records from a data container `orderunnest`:

```
orderunnest4 = limit orderunnest 4;
```

```
dump orderunnest4;
```

(17) How to save the results in HDFS ?

To save in HDFS the contents of a data container created by processing Pig Latin commands use a command `store` in the following way:

```
store ordergrp into '/user/bigdata/orders_dir' using
PigStorage('|');
store ordcube into '/user/bigdata/ordcube_dir' using
PigStorage('|');
```

Next, start Terminal and process the following commands to list the contents of HDFS:

```
$HADOOP_HOME/bin/hadoop fs -ls /user/bigdata/orders_dir
$HADOOP_HOME/bin/hadoop fs -cat /user/bigdata/orders_dir/part-r-00000
```

(18) How to process a Pig script ?

It is possible to process a sequence of commands separated with semicolons as a value of -e parameter of Pig command line interface. Use Terminal to process the following command at shell prompt:

```
$PIG_HOME/bin/pig -e "orders = load '/user/bigdata/orders.txt'
using PigStorage(','); dump orders;"
```

Of course it is possible to create a text file `script.pig` that contains all your Pig commands and process it as a script in the following way:

```
$PIG_HOME/bin/pig -f script.pig.
```

Appendix A

The outcomes of `help` command.

```
<pig latin statement>; - See the PigLatin manual for details:
http://hadoop.apache.org/pig
File system commands:
    fs <fs arguments> - Equivalent to Hadoop dfs command:
http://hadoop.apache.org/common/docs/current/hdfs_shell.html
Diagnostic commands:
    describe <alias>[::<alias] - Show the schema for the alias. Inner
aliases can be described as A::B.
    explain [-script <pigscript>] [-out <path>] [-brief] [-dot|-xml] [-
param <param_name>=<param_value>]
        [-param_file <file_name>] [<alias>] - Show the execution plan to
compute the alias or for entire script.
        -script - Explain the entire script.
        -out - Store the output into directory rather than print to stdout.
        -brief - Don't expand nested plans (presenting a smaller graph for
overview).
        -dot - Generate the output in .dot format. Default is text format.
        -xml - Generate the output in .xml format. Default is text format.
        -param <param_name - See parameter substitution for details.
```

```
        -param_file <file_name> - See parameter substitution for details.
        alias - Alias to explain.
    dump <alias> - Compute the alias and writes the results to stdout.
Utility Commands:
    exec [-param <param_name>=param_value] [-param_file <file_name>]
<script> -
        Execute the script with access to grunt environment including
aliases.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        script - Script to be executed.
    run [-param <param_name>=param_value] [-param_file <file_name>]
<script> -
        Execute the script with access to grunt environment.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        script - Script to be executed.
    sh  <shell command> - Invoke a shell command.
    kill <job_id> - Kill the hadoop job specified by the hadoop job id.
    set <key> <value> - Provide execution parameters to Pig. Keys and
values are case sensitive.
        The following keys are supported:
        default_parallel - Script-level reduce parallelism. Basic input
size heuristics used by default.
        debug - Set debug on or off. Default is off.
        job.name - Single-quoted name for jobs. Default is PigLatin:<script
name>
        job.priority - Priority for jobs. Values: very_low, low, normal,
high, very_high. Default is normal
        stream.skippath - String that contains the path. This is used by
streaming.
        any hadoop property.
    help - Display this message.
    history [-n] - Display the list statements in cache.
        -n Hide line numbers.
    quit - Quit the grunt shell.For a good start use a command help to get
a pretty comprehensive help from HBase command Line Interface (CLI). A
complete printout of help is listed at the end of this document.
```
