Lecture 5 – More Apache Pig

BDAT 1002

Complex Data Types in Pig

Introduction to Complex Data Types

- So far we have had simple data types
  - Float, int, etc
- But Pig can also support data types that are complex and nested
- In this section, we will learn about three complex data types in Pig
  - Tuple
  - Bag
  - Map

Introduction to Complex Data Types

- If you do a describe on grp_by_symbol, you'll see a funny structure
- This is a complex structure containing tuple and bag → see the text file for a formatted output

Tuple and Bag

- A **tuple** is just a row, a collection of columns
  - A tuple will have *round brackets*
- A **bag** is a collection of tuples
  - A bag will have *braces around it*

TUPLE ( )

(ABCSE,CASC,2003-12-22T00:00:00.000Z,22.02,22.2,21.94,22.09,36700,20.29)

BAG { }

{
(ABCSE,CATO,2003-10-08T00:00:00.000Z,22.48,22.5,22.01,22.06,92000,12.0),
(ABCSE,CATO,2003-10-09T00:00:00.000Z,21.3,21.59,21.16,21.45,373500,11.67),
.....
}

grp_by_symbol structure

- From the curly brackets, we know grp_by_sym is a bag
  - The first column is called "group" and it is of type character array
  - The second column is named "filter_by_yr" and is a bag
- The first column is named "group" it refers to the grouping column
  - It refers to symbol in this case

grp_by_symbol structure

- Let's output the first 10 records of grp_by_symbol

```
grunt> limit10= LIMIT grp_by_sym 10;

grunt> DUMP limit10;
```

- Output looks a bit cluttered, but format is shown in textfile

Nesting

- Note that we have a bag made up of two columns one of which is another bag
- This is called **nesting** and it is completely legal inside of Pig
- We also have a bag inside a tuple in this case
  - Again, an example of nested data type

Projecting Nested Columns

- Let's take a look at how to project the columns from grp_by_sym

```
grunt> close_by_sym = FOREACH grp_by_sym
GENERATE group, filter_by_yr.close;
```

- Display 10 records

```
grunt> limit10 = LIMIT close_by_sym 10;

grunt> DUMP limit10;
```

Projecting

- By having grouped records by symbols, we can do some meaningful aggregate operations → average, max, min
- So let's find out the maximum closing price per each symbol

```
grunt> max_close_by_sym = FOREACH grp_by_sym
GENERATE group, MAX(filter_by_yr.close);

grunt> limit10 = LIMIT close_by_sym BY 10;

grunt> DUMP limit10;
```

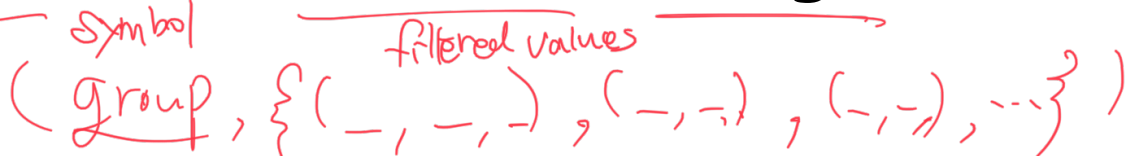
Grouping by more than one column

- Want to find out the maximum closing price for each symbol by year
- First we need to group our data set by symbol and year

```
grunt> grp_by_sym_yr = GROUP stocks BY (symbol,
GetYear(date));

grunt> DESCRIBE grp_by_sym_yr
```

- Note that grp_by_sym_yr has two columns, first is a tuple and second is a bag of records



Grouping by more than one column

- To find the maximum price by year

```
grunt> max_close_by_sym_yr = FOREACH grp_by_sym_yr
GENERATE group, MAX(stocks.close);

grunt> limit10 = LIMIT max_close_by_sym_yr 10;

grunt> DUMP limit10;
```

- Notice that the output is nested, a tuple inside a tuple

Complex type loading

- We can also just show individual stocks (without the year)

```
grunt> max_close_by_sym_yr = FOREACH grp_by_sym_yr
GENERATE group.symbol, group.$1, MAX(stocks.close);

grunt> limit10 = LIMIT max_close_by_sym_yr 10;

grunt> DUMP limit10;
```

- Notice that this output is not nested like th previous output

Complex Type Loading

Complex type loading

- Sometimes we may get a data set that is already in a format we can use
- For example, let's save the grp_by_sym_yr data to HDFS

```
grunt> STORE grp_by_sym_yr INTO
'/BDAT1002/grp_by_sym_yr';
```

- This data is made up of bags and tuples, so we should also be able to load it just as we did before
  - How do we do this?

Complex type loading

- Now take a look at the output

```
[cloudera@quickstart] hadoop fs -ls
/BDAT1002/grp_by_sym_yr

[cloudera@quickstart] hadoop fs -cat
/BDAT1002/grp_by_sym_yr/part-r-00000
```

Complex type loading

- Let's say we want to find the maximum volume by each symbol by year
- It's simple because our data is already grouped by symbol and year
- So we want to load the dataset but make sure that it is properly formatted

Complex type loading

- Let's look at the structure again from our output
  - We have two columns separated by TAB
  - First column has two values, separated by comma and enclosed in paranthesis → tuple
  - Second column is a bag
- Nothing ground breaking!
  - Point is you should be able to recognize complex types

Complex type loading

- For the tuple data type we will use "tuple (name1:type1, name2:type2, …)
- For bags, we will use data type "bag{name1:type1, nam2:type2, …}
- Using keywords bag and tuple are not necessary, you can simply use braces or round brackets to indicate the type

Complex type loading

- Here is our command

```
grunt> complex_type = LOAD
'/BDAT1002/grp_by_sym_yr' AS
(grp:tuple(sym:chararray, yr:int), stocks_b:bag
{stocks_t:tuple(exchange:chararray,
symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float, volume:int,
adj_close:float)}) ;
```

Complex type loading

- Here is our command

```
grunt> complex_type = LOAD
'/BDAT1002/grp_by_sym_yr' AS  (grp:(sym:chararray,
yr:int), stocks_b: {stocks_t:(exchange:chararray,
symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float, volume:int,
adj_close:float)}) ;
```

Complex type loading

- And to extract some information

```
grunt> max_vol_by_yr = FOREACH complex_type
GENERATE grp.sym, grp.yr, MAX(stocks_b.volume);

grunt> limit10 = LIMIT max_vol_by_yr 10;
grunt> DUMP limit10;
```

- Note that we do not need to do stocks_b.stocks_t.volume

Map Complex type

- Map is a collection of key-value pairs
- Maps have square brackets [ ]
- Each key-value pair is separated by a comma
- Key and value are separated by a #
- To work with maps, there is another dataset, save this to your HDFS and take a look at the output

Map Complex type

- Here is a few records in this dataset

```
328;ADMIN HEARNG;[street#939 W El Camino,city#Chicago,state#IL]
43;ANIMAL CONTRL;[street#415 N Mary Ave,city#Chicago,state#IL]
210;AVIATION;[street#2373 S Archer Ave,city#Chicago,state#IL]
```

- Every row has three pieces of information
  - Department ID
  - Name
  - Address
- The address column is a Map
  - Key value pair separated by hashtag
  - Pairs separated by comma

Loading a Map

- Use keyword map[] as data type

```
grunt> departments = LOAD
'/BDAT1002/pig/department_dataset' USING
PigStorage(';') AS (dept_id:int,
dept_name:chararray, address:map[]);
```

Projecting elements in a Map

- What if we want to get to the individual values in the key-value pair?
  - Use the hashtag qualifier

```
grunt> dept_addr = FOREACH departments GENERATE
dept_name, address#'street' AS street,
address#'city' AS city, address#'state' AS state;

grunt> DUMP dept_addr;
```

Projecting elements in a Map

- We learned:
  - Three complex datatypes in Pig
  - How to spot and work with complex data types

Pig Latin - Joins

Introduction

- Our datasets are often related to each other
- When this happens, it is natural for us to join datasets to create new relations
- One of the difficult things to do with traditional mapReduce is Joins
  - Using a level language like Java would take a lot of time and effort
- We want to see how to do Joins using Apache Pig

Joining Tables

- How to choose values from multiple tables?

Employee

ID	FirstName	LastName	HireDate	DepartmentID	...
734	Aaron	Cooper	4/17/09	2	
735	Lou	Donoghue	5/22/05	4	
736	Alice	Bailey	9/1/99	(null)	
737	Oswald	Hall	3/19/11	5	
738	John	Velasquez	4/5/10	4	
...	...				

Department

DepartmentID	Name	Location	BudgetCode	...
1	Production	CA	A4	...
2	R&D	AZ	B17	
3	Marketing	CA	A7	
4	Sales	CA	A7	
5	PR	UK	C9	
...				

```
SELECT FirstName, LastName, HireDate,
       DepartmentID
FROM Employee
```

FirstName	LastName	HireDate	DepartmentID
Aaron	Cooper	4/17/09	2
Lou	Donoghue	5/22/05	4
Alice	Bailey	9/1/99	(null)
Oswald	Hall	3/19/11	5
John	Velasquez	4/5/10	4
...			

Joining Tables

- Can use JOIN → somewhat unintuitive

Employee

ID	FirstName	LastName	HireDate	DepartmentID	...
734	Aaron	Cooper	4/17/09	2	
735	Lou	Donoghue	5/22/05	4	
736	Alice	Bailey	9/1/99	(null)	
737	Oswald	Hall	3/19/11	5	
738	John	Velasquez	4/5/10	4	
...	...				

Department

DepartmentID	Name	Location	BudgetCode	...
1	Production	CA	A4	...
2	R&D	AZ	B17	
3	Marketing	CA	A7	
4	Sales	CA	A7	
5	PR	UK	C9	
...				

```
SELECT FirstName, LastName, HireDate,
       Employee. DepartmentID , Name, Location
FROM Employee JOIN Department
ON Employee.DepartmentID = Department.DepartmentID
```

FirstName	LastName	HireDate	Employee. DepartmentID	Name	Location
Aaron	Cooper	4/17/09	2	R&D	AZ
Lou	Donoghue	5/22/05	4	Sales	CA
Oswald	Hall	3/19/11	5	PR	UK
John	Velasquez	4/5/10	4	Sales	CA
...					

Joining Tables

- Better to use INNER JOIN
  - Only bring values where there is a match in both tables
  - Same functionality as JOIN but more informative

Employee

ID	FirstName	LastName	HireDate	DepartmentID	...
734	Aaron	Cooper	4/17/09	2	
735	Lou	Donoghue	5/22/05	4	
736	Alice	Bailey	9/1/99	(null)	
737	Oswald	Hall	3/19/11	5	
738	John	Velasquez	4/5/10	4	
...	...				

Department

DepartmentID	Name	Location	BudgetCode	...
1	Production	CA	A4	...
2	R&D	AZ	B17	
3	Marketing	CA	A7	
4	Sales	CA	A7	
5	PR	UK	C9	
...				

```
SELECT FirstName, LastName, HireDate,
       Employee. DepartmentID , Name, Location
FROM Employee  INNER JOIN Department
ON Employee.DepartmentID = Department.DepartmentID
```

FirstName	LastName	HireDate	Employee. DepartmentID	Name	Location
Aaron	Cooper	4/17/09	2	R&D	AZ
Lou	Donoghue	5/22/05	4	Sales	CA
Oswald	Hall	3/19/11	5	PR	UK
John	Velasquez	4/5/10	4	Sales	CA
...					

Joining Tables

- OUTER JOIN
  - We make one table take precedence → I want to see all the rows in this table and show the matching data if possible
  - LEFT → Table on left hand side of JOIN
  - RIGHT→ Table on right hand side of JOIN takes precedence

Employee

ID	FirstName	LastName	HireDate	DepartmentID	...
734	Aaron	Cooper	4/17/09	2	
735	Lou	Donoghue	5/22/05	4	
736	Alice	Bailey	9/1/99	(null)	
737	Oswald	Hall	3/19/11	5	
738	John	Velasquez	4/5/10	4	
...	...				

Department

DepartmentID	Name	Location	BudgetCode	...
1	Production	CA	A4	...
2	R&D	AZ	B17	
3	Marketing	CA	A7	
4	Sales	CA	A7	
5	PR	UK	C9	
...				

```
SELECT FirstName, LastName, HireDate,
       Employee. DepartmentID , Name, Location
FROM Employee LEFT OUTER JOIN Department
ON Employee.DepartmentID = Department.DepartmentID
```

FirstName	LastName	HireDate	Employee. DepartmentID	Name	Location
Aaron	Cooper	4/17/09	2	R&D	AZ
Lou	Donoghue	5/22/05	4	Sales	CA
Alice	Bailey	9/1/99	(null)	(null)	(null)
Oswald	Hall	3/19/11	5	PR	UK
John	Velasquez	4/5/10	4	Sales	CA
...					

Joining Tables

- OUTER JOIN
  - We make one table take precedence → I want to see all the rows in this table and show the matching data if possible
  - LEFT → Table on left hand side of JOIN
  - RIGHT→ Table on right hand side of JOIN takes precedence

Employee

ID	FirstName	LastName	HireDate	DepartmentID	...
734	Aaron	Cooper	4/17/09	2	
735	Lou	Donoghue	5/22/05	4	
736	Alice	Bailey	9/1/99	(null)	
737	Oswald	Hall	3/19/11	5	
738	John	Velasquez	4/5/10	4	
...	...				

Department

DepartmentID	Name	Location	BudgetCode	...
1	Production	CA	A4	...
2	R&D	AZ	B17	
3	Marketing	CA	A7	
4	Sales	CA	A7	
5	PR	UK	C9	
...				

```
SELECT FirstName, LastName, HireDate,
       Employee. DepartmentID , Name, Location
FROM Employee RIGHT OUTER JOIN Department
ON Employee.DepartmentID = Department.DepartmentID
```

FirstName	LastName	HireDate	Employee. DepartmentID	Name	Location
(null)	(null)	(null)	(null)	Production	CA
Aaron	Cooper	4/17/09	2	R&D	AZ
Lou	Donoghue	5/22/05	4	Sales	CA
Oswald	Hall	3/19/11	5	PR	UK
John	Velasquez	4/5/10	4	Sales	CA
(null)	(null)	(null)	(null)	Marketing	CA

Our second dataset

- As you can see, to do joins we need to data sets that are related
- We are going to use a dividends data set
- Download and load this to your HDFS

Loading Datasets

- Start a new grunt shell
- Load the stock dataset

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',') AS (exchange:chararray,
symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float, volume:int,
adj_close:float);
```

Loading Datasets

- Load dividends dataset

```
grunt> divs = LOAD '/BDAT1002/dividends' USING
PigStorage(',') AS (exchange:chararray,
symbol:chararray, date:datetime, dividends:float);
```

First Join

- Remember our stock dataset had daily stock trading information
- Dividend is paid on certain days
- What if we want to know the stock information on only the days the dividend is being paid?
  - Classic inner join
  - Symbol and date are common to both datasets

First Join

- Here is the instruction to do an inner join

```
grunt> join_inner = JOIN stocks BY (symbol, date) ,
divs BY (symbol, date);
```

First Join

- Now the join is easy, but now how do we use our new relation?
  - Or how do we project columns from our new table/relation?
- If you are not sure do a describe, all the information you need will be given

```
grunt> DESCRIBE join_inner;
```

First Join

- join_inner has columns from both the stocks dataset and the dividends dataset
  - Columns from stocks are given as **stocks::**
  - Columns from dividends are given as **divs::**
- So it is easy for us to get specific columns

```
grunt> join_project  = FOREACH join_inner GENERATE
stocks::symbol, divs::date, divs::dividends;
```

Left outer join

- Left joins take all data records from the left hand dataset and combines them with the records from the right hand side
- If there is no match for a record, all the data columns for the right side will be NULL

```
grunt> join_left = JOIN stocks BY (symbol, date)
LEFT OUTER, divs BY (symbol, date);
```

Left outer join

- If you want to filter out the recods that do not have a matching dividend record, you can use the following command:

```
grunt> filterleftjoin = FILTER join_left BY
divs::symbol IS NOT NULL;
```

Right Outer Join

- Right joins take all data records from the right hand dataset and combines them with the records from the left hand side
- If there is no match for a record, all the data columns for the left side will be NULL

```
grunt> join_right = JOIN stocks BY (symbol, date)
RIGHT OUTER, divs BY (symbol, date);
```

Full Outer Join

- Combination of left and right outer joins
- Get all the data in your dataset in one table
- If no matches, substitute NULL

```
grunt> join_full = JOIN stocks BY (symbol, date)
FULL, divs BY (symbol, date);
```

Interview Question

- So far our joins are based on equality condition
- What about none equality condition?
  - Example: inner join between stocks and dividends when the symbol for the stocks dataset DOES NOT match the dividends dataset

Answer

- None equality joins are very difficult to do in MapReduce
- Therefore, no direct way to do in PigLatin

Follow up Interview Question

- How do you work around that problem?
- Answer
  - Do a cross join and use the filter command

CROSS command

```
grunt> crs = CROSS stocks, divs;
```

- This will cross your dataset meaning, if you have 10 records in stocks and 10 records in divs, you'll get 100 records

CROSS command

```
grunt> crs = CROSS stocks, divs;
grunt> non_equi = FILTER crs by stocks::symbol !=
divs::symbol;
```

- This will cross your dataset meaning, if you have 10 records in stocks and 10 records in divs, you'll get 100 records
  - So be careful with CROSS operations
  - Can be *very expensive!*

Multi-way joins

- So far our joins have only two tables/relations
- Can we do joins with multiple tables?
- Yes but only with inner joins
  - And join key has to be the same for all tables
- First we need a new dataset, download and save to HDFS the ***company*** dataset

Companies dataset Structure

```
HU;HU Inc.;[street#98 N Wells Ave,city#San Jose,state#CA]
IJB;IJB Inc.;[street#101 S Draper Ave,city#Miami,state#FL]
```

```
companies = LOAD '/BDAT1002/companies' USING
PigStorage(';’) AS (symbol:chararray,
name:chararray, address: map[]);

cmp = FOREACH companies GENERATE symbol, name,
address#'street', address#'city', address#'state';
```

Multi-way joins

- Invalid join

```
grunt> join_multi = JOIN stocks BY (symbol, date),
divs BY (symbol, date), cmp BY symbol;
```

- Valid Join

```
grunt> join_multi = JOIN stocks BY symbol, divs BY
symbol, cmp BY symbol;
```

Interview Question

- How do you do a join like operation without using a JOIN operator?

```
grunt> cgrp = COGROUP stocks BY (symbol, date),
divs BY (symbol, date);
```

- Similar to a OUTER JOIN, except that it results in a nested output
  - Whereas outer join results in a flat output

Interview Question

- To get something similar to an INNER JOIN, we can use the following:

```
grunt> filter_empty_divs = FILTER cgrp BY (NOT
IsEmpty(stocks)) AND  (NOT IsEmpty(divs));
```

Review Questions

Review Questions

- [T/F] Pig has to be installed on all nodes

Review Questions

- Assume you have a Pig relation with 10 columns. What happens when you load a dataset with 12 columns in each row?
  - Pig will load the dataset with no issues
  - Pig load instruction will cause an error

Review Questions

- [T/F] A dataset can be loaded to Pig without specifying the schema

Review Questions

- What will be the datatype of the columns when you don't specify a schema loading your dataset?

Review Questions

- Which of the below is not a Pig type?
  - String
  - Int
  - Bag
  - None of the above

Review Questions

- Which of the below is not possible?
  - Tuple inside a bag
  - Bag inside a Tuple
  - Tuple inside a Bag inside a Tuple
  - None of the above

Review Questions

- It is not possible to join more than 2 relations in Pig

Review Questions

- Multiway join is only possible with Inner joins and not with any other join

Review Questions

- Which operator in Pig can be used to group when there is more than one relation involved?
  - GROUP
  - MULTIGROUP
  - COGROUP
  - None of the above