CC5212-1

Procesamiento Masivo de Datos

Otoño 2019

Lecture 4

Apache Pig

Aidan Hogan

aidhog@gmail.com

Hadoop: Wrapping Up

Hadoop: Supermarket Example

ReceiptItems

Receipt ID	Item ID
R1401	I306
R1401	I306
R1401	I504
R1402	I007
R1402	I306
R1403	I306
R1403	I504
...	...

ReceiptTimes

Receipt ID	Time
R1403	19:00
R1401	18:59
R1402	19:01
...	...

ItemDetails

Item ID	Name	Price ($)
I306	Zanahoria 500g	500
I504	CocaCola 3L	1400
I007	Comfort	1200
...	...	

Compute total sales per hour of the day?

Output

Hour	Total
...	...
18:00–18:59	$2400
19:00–19:59	$3600
...	...

More in Hadoop: Multiple Inputs

```
public class RevenuePerHour {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 4) {
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");
            System.exit(2);
        }

        Job job1 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),
                TextInputFormat.class, ReceiptItemsMapper.class);
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),
                TextInputFormat.class, ReceiptTimesMapper.class);
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));

        job1.setReducerClass(ItemsTimesReducer.class);
        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(Text.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(Text.class);
        job1.waitForCompletion(true);
```

Multiple inputs, different map for each

One reducer

More in Hadoop: Chaining Jobs

```
public class RevenuePerHour {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 4) {
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");
            System.exit(2);
        }

        Job job1 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),
                TextInputFormat.class, ReceiptItemsMapper.class);
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),
                TextInputFormat.class, ReceiptTimesMapper.class);
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));

        job1.setReducerClass(ItemsTimesReducer.class);
        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(Text.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(Text.class);
        job1.waitForCompletion(true);

        Job job2 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job2, new Path(otherArgs[2]),
                TextInputFormat.class, ItemsTimesMapper.class);
        MultipleInputs.addInputPath(job2, new Path(otherArgs[3]),
                TextInputFormat.class, ItemsPricesMapper.class);
        FileOutputFormat.setOutputPath(job2, new Path(otherArgs[4]));

        job2.setReducerClass(TimesPricesReducer.class);
        job2.setMapOutputKeyClass(LongWritable.class);
        job2.setMapOutputValueClass(Text.class);
```

Run and wait

Output of Job1 set to Input of Job2

More in Hadoop: Number of Reducers

```
job.setNumReduceTasks(1);
```

Set number of parallel reducer tasks for the job

Why would we ask for 1 reduce task?

Output requires a merge on one machine (for example, sorting, top-*k*)

Hadoop: Filtered Supermarket Example

ReceiptItems

Receipt ID	Item ID
R1401	I306
R1401	I306
R1401	I504
R1402	I007
R1402	I306
R1403	I306
R1403	I504
...	...

ReceiptTimes

Receipt ID	Time
R1403	19:00
R1401	18:59
R1402	19:01
...	...

ItemDetails

Item ID	Name	Price ($)
I306	Zanahoria 500g	500
I504	CocaCola 3L	1400
I007	Comfort	1200
...	...	

Compute total sales per hour of the day ...
but exclude certain item IDs passed as an input file?

Output

Hour	Total
...	...
18:00–18:59	$1400
19:00–19:59	$2600
...	...

More in Hadoop: Distributed Cache

- Some tasks need “global knowledge”
  - Hopefully not too much though
- Use a distributed cache:
  - Makes global data available locally to all nodes
    - On the local hard-disk of each machine

How might we use this?

Make the filtered products global and read them (into memory?) when processing items

Apache Hadoop … Internals (if interested)

Apache Hadoop (MapReduce) Internals - Diagram

Fork me on GitHub

This project contains several diagrams describing **Apache Hadoop** internals (2.3.0 or later). Even if these diagrams are NOT specified in any formal or unambiguous language (e.g., UML), they should be reasonably understandable (here some **diagram notation conventions**) and useful for any person who want to grasp the main ideas behind Hadoop. Unfortunately, not all the internal details are covered by these diagrams. You are free to help :)

Hadoop Internals (2.3.0 or later) from Emilio Coppa

http://ercoppa.github.io/HadoopInternals/

HADOOP VS. SQL

Hadoop: (ಠ_ಠ)

```
public class RevenuePerHour {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 4) {
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");
            System.exit(2);
        }

        Job job1 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),
                TextInputFormat.class, ReceiptItemsMapper.class);
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),
                TextInputFormat.class, ReceiptTimesMapper.class);
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));

        job1.setReducerClass(ItemsTimesReducer.class);
        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(Text.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(Text.class);
        job1.waitForCompletion(true);

        Job job2 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job2, new Path(otherArgs[2]),
                TextInputFormat.class, ItemsTimesMapper.class);
        MultipleInputs.addInputPath(job2, new Path(otherArgs[3]),
                TextInputFormat.class, ItemsPricesMapper.class);
        FileOutputFormat.setOutputPath(job2, new Path(otherArgs[4]));

        job2.setReducerClass(TimesPricesReducer.class);
        job2.setMapOutputKeyClass(LongWritable.class);
        job2.setMapOutputValueClass(Text.class);
```

SQL

```
SELECT C1.actor AS a1, C2.actor AS a2, COUNT(C1.m_name) AS num
FROM CastMovie C1, CastMovie C2
WHERE C1.m_name = C2.m_name
  AND C2.m_year = C2.m_year
  AND C1.m_id = C2.m_id
  AND C1.m_type = 'THEATRICAL_MOVIE'
  AND C1.actor < C2.actor
GROUP BY C1.actor, C2.actor
ORDER BY num DESC
```

So why not just use SQL?

Relational database engines not typically built for large workloads over bulk data; they optimise for answering queries that touch a small fraction of the data.

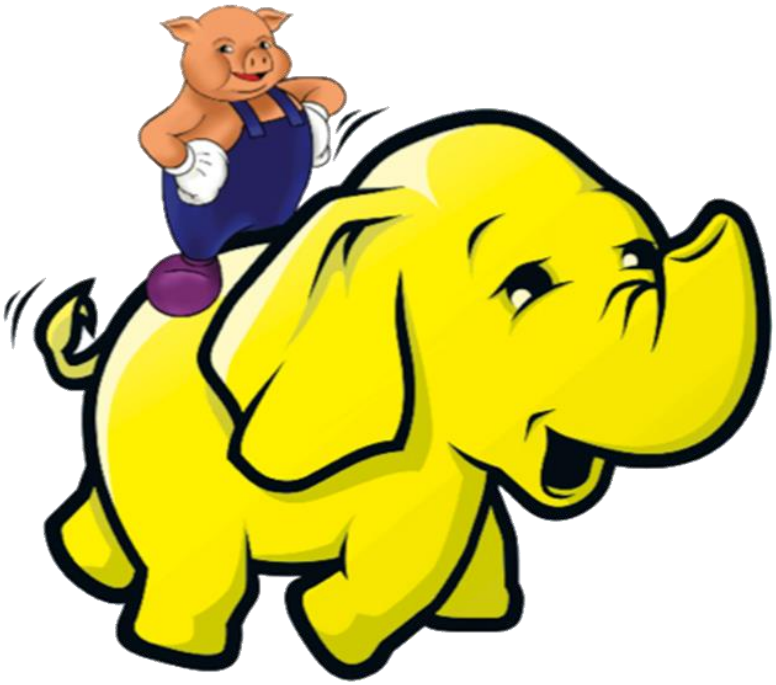
At some stage, they will not scale further.

But this is a reason not to use a relational database.
The question was: why not just use SQL?

Apache Pig: Overview

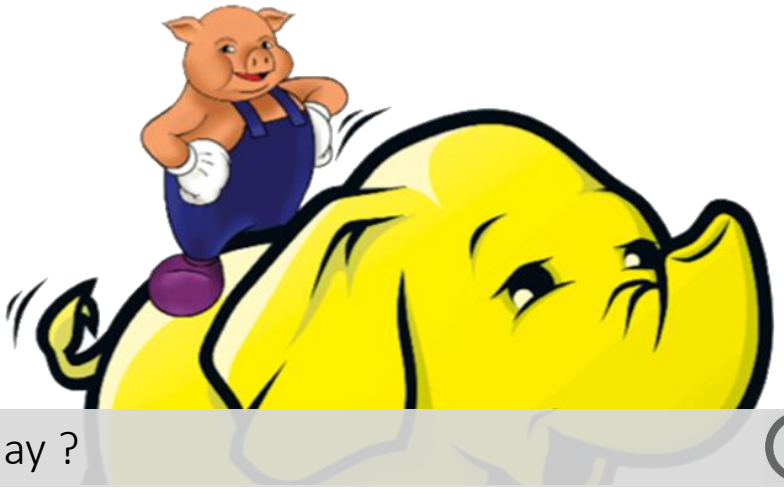
Apache Pig

- Create MapReduce programs to run on Hadoop
- Use a high-level "scripting" language called Pig Latin
- Can embed User Defined Functions: call a Java function (or Python, Ruby, etc.)
- Based on Pig Relations



Apache Pig

- Create MapReduce programs to run on Hadoop
- Use a high-level "scripting" language called Pig Latin
- Can embed Functions: c (or Python,
- Based on Pi



Atwhay anguagelay isyay isthay ?

pig Lat·in

/ˈpig ˌlatn/

noun

a made-up language formed from English by transferring the initial consonant or consonant cluster of each word to the end of the word and adding a vocalic syllable (usually ˈpig ˌlatn: so *chicken soup* would be translated to *ickenchay oupsay*. Pig Latin is typically spoken playfully, as if to convey secrecy.

Translations, word origin, and more definitions

Pig Latin: Hello Word Count

```
input_lines = LOAD '/tmp/book.txt' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;
```

Map

Reduce

```
-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;
```

```
-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
```

Map + Reduce

```
STORE ordered_word_count INTO '/tmp/book-word-count.txt';
```

Any ideas which lines correspond to map and which to reduce?

Apache Pig: An Example

Pig: Products by Hour

transact.txt

```
customer412     1L_Leche          2014-03-31T08:47:57Z     $900
customer412     Nescafe           2014-03-31T08:47:57Z     $2.000
customer412     Nescafe           2014-03-31T08:47:57Z     $2.000
customer413     400g_Zanahoria    2014-03-31T08:48:03Z     $1.240
customer413     El_Mercurio       2014-03-31T08:48:03Z     $500
customer413     Gillette_Mach3    2014-03-31T08:48:03Z     $8.250
customer413     Santo_Domingo     2014-03-31T08:48:03Z     $2.450
customer413     Nescafe           2014-03-31T08:48:03Z     $2.000
customer414     Rosas             2014-03-31T08:48:24Z     $7.000
customer414     Chocolates        2014-03-31T08:48:24Z     $9.230
customer414     300g_Frutillas    2014-03-31T08:48:24Z     $1.230
customer415     Nescafe           2014-03-31T08:48:35Z     $2.000
customer415     12 Huevos         2014-03-31T08:48:35Z     $2.200
…
```

Find the number of items sold per hour of the day

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
```

User-defined-functions written in Java (or Python, Ruby, etc. …)

userDefinedFunctions.jar

```
public class ExtractHour extends EvalFunc<String> {
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try{
            String timestamp = (String)input.get(0);
            return timestamp.substring(6, 8);
        }catch(Exception e){
            System.err.println("ExtractHour: failed to proces input; error - " + e.getMessage());
            return null;
        }
    }
```

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
```

View data as a (streaming) relation with fields (cust, item, etc.) and tuples (data rows) …

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	$900
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	$1.240
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
```

Filter tuples depending on their value for a given attribute (in this case, price < 1000)

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	$900
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	$1.240
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
```

Filter tuples depending on their value for a given attribute (in this case, price < 1000)

premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	$1.240
customer413	Gillette_Mach3	2014-03-31T08:48:03Z	$8.250
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
```

premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	$1.240
customer413	Gillette_Mach3	2014-03-31T08:48:03Z	$8.250
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
```

hourly:

cust	item	hour	price
customer412	Nescafe	08	$2.000
customer412	Nescafe	08	$2.000
customer413	400g_Zanahoria	08	$1.240
customer413	Gillette_Mach3	08	$8.250
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
```

hourly:

cust	item	hour	price
customer412	Nescafe	08	$2.000
customer412	Nescafe	08	$2.000
customer413	400g_Zanahoria	08	$1.240
customer413	Gillette_Mach3	08	$8.250
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
```

unique:

cust	item	hour	price
customer412	Nescafe	08	$2.000
customer413	400g_Zanahoria	08	$1.240
customer413	Gillette_Mach3	08	$8.250
customer413	Santo_Domingo	08	$2.450
…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
```

hrItem:

[item,hour]	cust	item	hour	price
[Nescafe,08]	customer412	Nescafe	08	$2.000
	customer413	Nescafe	08	$2.000
	customer415	Nescafe	08	$2.000
[400g_Zanahoria,08]	customer413	400g_Zanahoria	08	$1.240
…	…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
```

hrItem:

[item,hour]	cust	item	hour	price
[Nescafe,08]	customer412	Nescafe	08	$2.000
	customer413	Nescafe	08	$2.000
count	customer415	Nescafe	08	$2.000
[400g_Zanahoria,08]	customer413	400g_Zanahoria	08	$1.240
…	…	…	…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
```

hrItemCnt:

[item,hour]	count
[400g_Zanahoria,08]	1
[Nescafe,08]	3
...	...

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
grunt> hrItemCntSorted = ORDER hrItemCnt BY count DESC;
```

hrItemCnt:

[item,hour]	count
[400g_Zanahoria,08]	1
[Nescafe,08]	3
...	...

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
grunt> hrItemCntSorted = ORDER hrItemCnt BY count DESC;
```

hrItemCntSorted:

[item,hour]	count
[Nescafe,08]	3
[400g_Zanahoria,08]	1
…	…

Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY org.udf.MinPrice1000(price);
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
grunt> hrItem = GROUP unique BY (item, hour);
grunt> hrItemCnt = FOREACH hrItem GENERATE flatten($0), COUNT($1) AS count;
grunt> hrItemCntSorted = ORDER hrItemCnt BY count DESC;
grunt> STORE hrItemCntSorted INTO 'output.txt';
```

hrItemCntSorted:

[item,hour]	count
[Nescafe,08]	3
[400g_Zanahoria,08]	1
…	…

Apache Pig: Schema

Pig Relations

- Pig Relations: Like relational tables
  - Except tuples can be “jagged”
  - Fields in the same column don’t need to be same type
  - Relations are by default unordered

- Pig Schema: Names for fields, etc.

```
… AS (cust, item, time, price);
```

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	$900
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	$1.240
…	…	…	…

Pig Fields

More readable!

- Pig Fields:
  - Reference using name
    - `premium = FILTER raw BY org.udf.MinPrice1000(price);`
  - … or position
    - `premium = FILTER raw BY org.udf.MinPrice1000($3);`

Starts at zero.

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	$900
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer412	Nescafe	2014-03-31T08:47:57Z	$2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	$1.240
…	…	…	…

Apache Pig: Types

Pig Simple Types

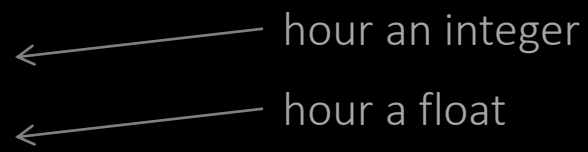
- Pig Types:
  - **LOAD** ‘transact.txt' **USING** PigStorage('\t') **AS** (cust:charArray, item:charArray, time:datetime, price:int);

- int, long, float, double, biginteger, bigdecimal, boolean, chararray (string), bytearray (blob), datetime

Pig Types: Duck Typing

- What happens if you omit types?
  - Fields default to bytearray
  - Implicit conversions if needed (~duck typing)

```
A = LOAD 'data' AS (cust, item, hour, price);
B = FOREACH A GENERATE hour + 4 % 24;
C = FOREACH A GENERATE hour + 4f % 24;
```



Pig Complex Types: Tuple

```
cat data;
(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int,t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;
((3,8,9),(4,5,6)) ((1,4,7),(3,7,5)) ((2,5,8),(9,5,8))

X = FOREACH A GENERATE t1.t1a,t2.$0;
```

A:

t1			t2		
t1a	t1b	t1c	t2a	t2b	t2c
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	5	8

Pig Complex Types: Tuple

```
cat data;
(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int,t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;
((3,8,9),(4,5,6)) ((1,4,7),(3,7,5)) ((2,5,8),(9,5,8))

X = FOREACH A GENERATE t1.t1a,t2.$0;
DUMP X;
(3,4)
(1,3)
(2,9)
```

X:

$0	$1
3	4
1	3
2	9

Pig Complex Types: Bag

```
cat data;
(3,8,9)
(2,3,6)
(1,4,7)
(2,5,8)

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
```

A:

c1	c2	c3
3	8	9
2	3	6
1	4	7
2	5	8

Pig Complex Types: Bag

```
cat data;
(3,8,9)
(2,3,6)
(1,4,7)
(2,5,8)

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
DUMP B;
(1,{(1,4,7)})
(2,{(2,5,8),(2,3,6)})
(3,{(3,8,9)})
```

B:

group (c1)	A		
	c1	c2	c3
3	3	8	9
2	2	3	6
	2	5	8
1	1	4	7

Pig Complex Types: Map

```
cat prices;
[Nescafe#”$2.000”]
[Gillette_Mach3#”$8.250”]

A = LOAD ‘prices’ AS (M:map []);
```

Pig Complex Types: Summary

- tuple: A row in a table / a list of fields
  - e.g., (customer412, Nescafe, 08, $2.000)
- bag: A set of tuples (allows duplicates)
  - e.g., { (cust412, Nescafe, 08, $2.000), (cust413, Gillette_Mach3, 08, $8.250) }
- map: A set of key–value pairs
  - e.g., [Nescafe#$2.000]

Apache Pig: Unnesting (Flatten)

Pig Latin: Hello Word Count

```
input_lines = LOAD '/tmp/book.txt' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;

STORE ordered_word_count INTO '/tmp/book-word-count.txt';
```

Pig Complex Types: Flatten Tuples

```
cat data;
(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int,t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;
((3,8,9),(4,5,6))
((1,4,7),(3,7,5))
((2,5,8),(9,5,8))

X = FOREACH A GENERATE flatten(t1), flatten(t2);
```

A:

t1			t2		
t1a	t1b	t1c	t2a	t2b	t2c
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	5	8

Pig Complex Types: Flatten Tuples

```
cat data;
(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int,t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;
((3,8,9),(4,5,6))
((1,4,7),(3,7,5))
((2,5,8),(9,5,8))

X = FOREACH A GENERATE flatten(t1), flatten(t2);
DUMP X;
(3,8,9,4,5,6)
(1,4,7,3,7,5)
(2,5,8,9,5,8)
```

X:

t1a	t1b	t1c	t2a	t2b	t2c
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	5	8

Pig Complex Types: Flatten Tuples

```
cat data;
(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int,t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;
((3,8,9),(4,5,6))
((1,4,7),(3,7,5))
((2,5,8),(9,5,8))

Y = FOREACH A GENERATE t1, flatten(t2);
```

A:

t1			t2		
t1a	t1b	t1c	t2a	t2b	t2c
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	5	8

Pig Complex Types: Flatten Tuples

```
cat data;
(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int,t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;
((3,8,9),(4,5,6))
((1,4,7),(3,7,5))
((2,5,8),(9,5,8))

Y = FOREACH A GENERATE t1, flatten(t2);
DUMP Y;
((3,8,9),4,5,6)
((1,4,7),3,7,5)
((2,5,8),9,5,8)
```

Y:

t1			t2a	t2b	t2c
t1a	t1b	t1c			
3	8	9	4	5	6
1	4	7	3	7	5
2	5	8	9	5	8

Pig Complex Types: Bag

```
cat data;
(3,8,9)
(2,3,6)
(1,4,7)
(2,5,8)

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
DUMP B;
(1,{(1,4,7)})
(2,{(2,5,8),(2,3,6)})
(3,{(3,8,9)})

C = FOREACH B GENERATE flatten(A);
```

B:

group (c1)	A		
	c1	c2	c3
3	3	8	9
2	2	3	6
	2	5	8
1	1	4	7

Pig Complex Types: Bag

```
cat data;
(3,8,9)
(2,3,6)
(1,4,7)
(2,5,8)

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
DUMP B;
(1,{(1,4,7)})
(2,{(2,5,8),(2,3,6)})
(3,{(3,8,9)})

C = FOREACH B GENERATE flatten(A);
DUMP C;
(3,8,9)
(2,3,6)
(2,5,8)
(1,4,7)
```

C:

c1	c2	c3
3	8	9
2	3	6
2	5	8
1	4	7

Pig Complex Types: Bag

```
cat data;
(3,8,9)
(2,3,6)
(1,4,7)
(2,5,8)

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
DUMP B;
(1,{(1,4,7)})
(2,{(2,5,8),(2,3,6)})
(3,{(3,8,9)})

D = FOREACH B GENERATE group, flatten(A);
```

B:

group (c1)	A		
	c1	c2	c3
3	3	8	9
2	2	3	6
	2	5	8
1	1	4	7

Pig Complex Types: Bag

```
cat data;
(3,8,9)
(2,3,6)
(1,4,7)
(2,5,8)

A = LOAD 'data' AS (c1:int, c2:int, c3:int);
B = GROUP A BY c1;
DUMP B;
(1,{(1,4,7)})
(2,{(2,5,8),(2,3,6)})
(3,{(3,8,9)})

D = FOREACH B GENERATE group, flatten(A);
DUMP D;
(3,3,8,9)
(2,2,3,6)
(2,2,5,8)
(1,1,4,7)
```

D:

group	c1	c2	c3
3	3	8	9
2	2	3	6
2	2	5	8
1	1	4	7

Apache Pig: Operators

Pig Atomic Operators

- Comparison
  ==, !=, >, <, >=, <=, `matches` (regex)
- Arithmetic
  + , –, *, /
- Reference
  tuple.field, map#value
- Boolean
  AND, OR, NOT
- Casting

Pig Conditionals

- Ternary operator:

```
hr12 = FOREACH item GENERATE hour%12, (hour>12 ? 'pm' : 'am');
```

- Cases:

```
X = FOREACH A GENERATE hour%12, (
      CASE
            WHEN hour>12 THEN 'pm'
            ELSE 'am'
      END
);
```

Pig Aggregate Operators

Can GROUP multiple items or COGROUP single item (COGROUP considered more readable for multiple items)

- Grouping:
  - GROUP: group on a single relation
    - **GROUP** premium **BY** (item, hour);
  - COGROUP: group multiple relations
    - **COGROUP** premium **BY** (item, hour), cheap **BY** (item, hour);
- Aggregate Operations:
  - AVG, MIN, MAX, SUM, COUNT, SIZE, CONCAT

Pig Joins

```
cat data1;
(Nescafe,08,120)
(El_Mercurio,08,142)
(Nescafe,09,153)

cat data2;
(2000,Nescafe)
(8250, Gillette_Mach3)
(500, El_Mercurio)

A = LOAD 'data1' AS (prod:charArray, hour:int, count:int);
B = LOAD 'data2' AS (price:int, name:charArray);
X = JOIN A BY prod, B BY name;

DUMP X:
(El_Mercurio,08,142, 500, El_Mercurio)
(Nescafe,08,120, 2000,Nescafe)
(Nescafe,09,153, 2000,Nescafe)
```

X:

prod	hour	count	price	name
Nescafe	08	120	2000	Nescafe
Nescafe	09	153	2000	Nescafe
El_Mercurio	08	142	500	El_Mercurio

Pig Joins

- Inner join: As shown (default)
- Self join: Copy an alias and join with that
- Outer joins:
  - LEFT / RIGHT / FULL
- Cross product:
  - CROSS

Anyone remember what an INNER JOIN is versus an OUTER JOIN / LEFT / RIGHT / FULL versus a CROSS PRODUCT?

Pig Aggregate/Join Implementations

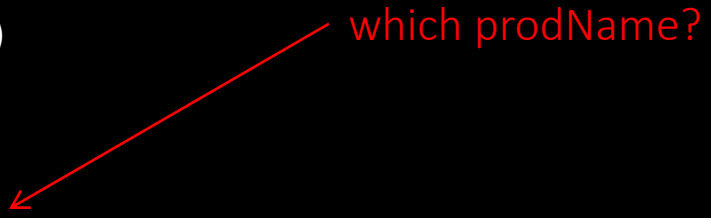
- Custom partitioning / number of reducers:
  - PARTITION BY specifies a UDF for partitioning
  - PARALLEL specifies number of reducers

```
X = JOIN A BY prod, B BY name PARTITION BY org.udp.Partitioner PARALLEL 5;
```

```
X = GROUP A BY hour PARTITION BY org.udp.Partitioner PARALLEL 5;
```

Pig: Disambiguate

```
cat data1;
(Nescafe,08,120)
(El_Mercurio,08,142)
(Nescafe,09,153)

cat data2;
(2000,Nescafe)
(8250,Gillette_Mach3)
(500,El_Mercurio)

A = LOAD 'data1' AS (prodName:charArray, hour:int, count:int);
B = LOAD 'data2' AS (price:int, prodName:charArray);
X = JOIN A BY prodName, B BY prodName;

DUMP X:
(El_Mercurio,08,142,500,El_Mercurio)
(Nescafe,08,120, 2000,Nescafe)
(Nescafe,09,153, 2000,Nescafe)

Y = FOREACH X GENERATE prodName
Y = FOREACH X GENERATE A::prodName
```



Pig: Split

```
raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
numeric = FOREACH raw GENERATE cust item time org.udf.RemoveDollarSign(price) AS price;
SPLIT numeric INTO cheap IF price<1000, premium IF price>=1000;
```

numeric:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
…	…	…	…

cheap:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
…	…	…	…

premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
…	…	…	…

Pig: Rank

```
raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
numeric = FOREACH raw GENERATE cust item time org.udf.RemoveDollarSign(price) AS price;
ranked = RANK numeric;
```

numeric:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
…	…	…	…

ranked:

rank	cust	item	time	price
1	customer412	1L_Leche	2014-03-31T08:47:57Z	900
2	customer412	Nescafe	2014-03-31T08:47:57Z	2.000
3	customer412	Nescafe	2014-03-31T08:47:57Z	2.000
4	customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
…	…	…	…	…

Pig: Rank

```
raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
numeric = FOREACH raw GENERATE cust item time org.udf.RemoveDollarSign(price) AS price;
ranked = RANK numeric BY price ASC, cust DESC;
```

numeric:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer412	Nescafe	2014-03-31T08:47:57Z	2.000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
…	…	…	…

ranked:

rank	cust	item	time	price
1	customer412	1L_Leche	2014-03-31T08:47:57Z	900
2	customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1.240
3	customer412	Nescafe	2014-03-31T08:47:57Z	2.000
3	customer412	Nescafe	2014-03-31T08:47:57Z	2.000
…	…	…	…	…

Pig: Other Operators

- FILTER: Filter tuples by an expression
- LIMIT: Only return a certain number of tuples
- MAPREDUCE: Run a native Hadoop .jar
- ORDER BY: Sort tuples
- SAMPLE: Sample tuples
- UNION: Concatenate two relations

Apache Pig: Nulos

Pig: Nulls

- Nulls represent incomplete information

```
cat data1;
(Nescafe,08,)
(El_Mercurio,08,142)
(,09,153)

A = LOAD 'data1' AS (prodName:charArray, hour:int, count:int);

DUMP A :
(Nescafe,08,)
(El_Mercurio,08,142)
(,09,153)
```

Pig: Nulls with JOIN

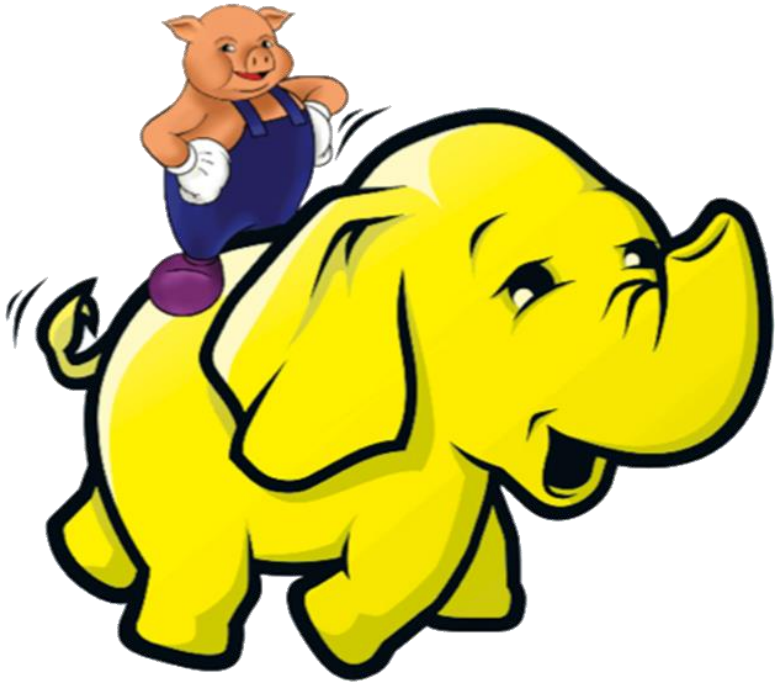
- Nulls represent incomplete information
- They behave as per nulls in SQL

```
cat data1;
(Nescafe,08,)
(El_Mercurio,08,142)
(,09,153)

cat data2;
(2000,)
(8250,Gillette_Mach3)
(500,El_Mercurio)

A = LOAD 'data1' AS (prodName:charArray, hour:int, count:int);
B = LOAD 'data2' AS (price:int, prodName:charArray);
X = JOIN A BY prodName, B BY prodName;

DUMP X:
(El_Mercurio,08,142,500,El_Mercurio)
```

Apache Pig: Execution

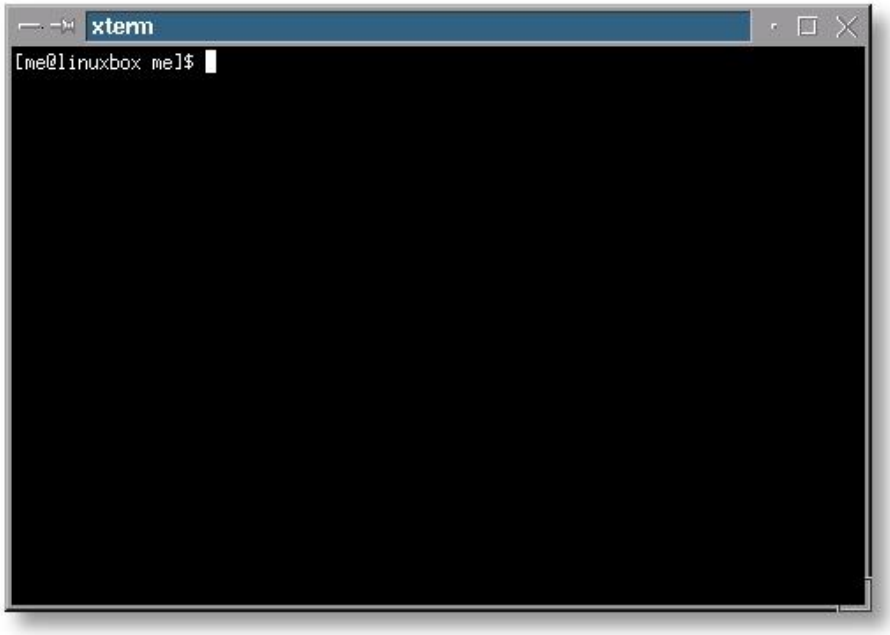
Pig translated to MapReduce in Hadoop

- Pig is only an interface/scripting language for MapReduce



Three Ways to Execute Pig: (i) Grunt

```
grunt> in_lines = LOAD '/tmp/book.txt' AS (line:chararray);
grunt> words = FOREACH in_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
grunt> filtered_words = FILTER words BY word MATCHES '\\w+';
grunt> …
…
grunt> STORE ordered_word_count INTO '/tmp/book-word-count.txt';
```



Three Ways to Execute Pig: (ii) Script

```
grunt> pig wordcount.pig
```

wordcount.pig

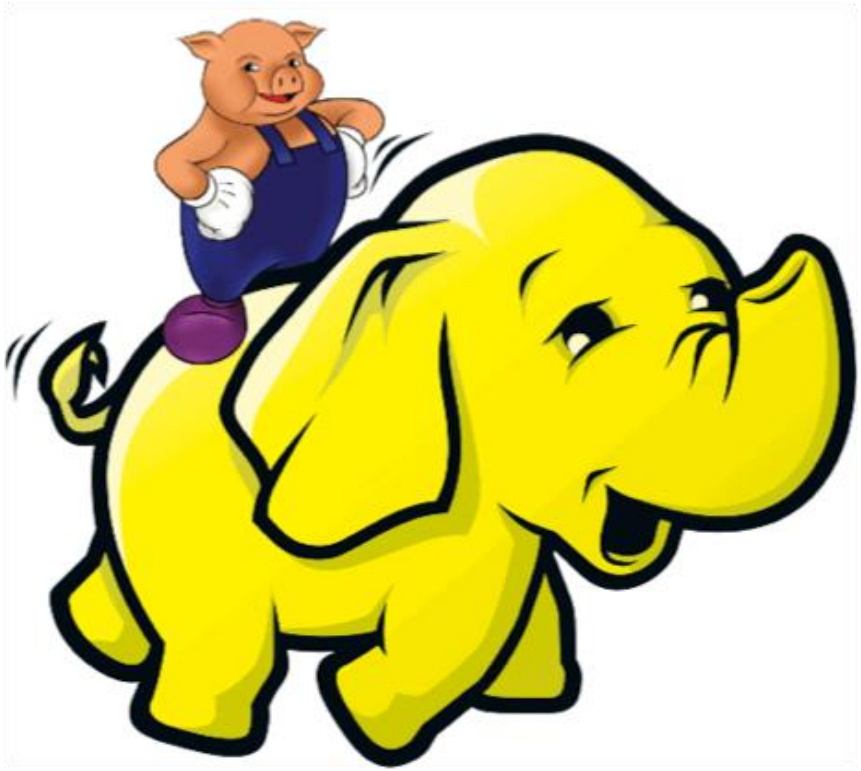
```
input_lines = LOAD '/tmp/book.txt' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;

STORE ordered_word_count INTO '/tmp/book-word-count.txt';
```

Three Ways to Execute Pig: (iii) Embedded

```java
package scratch;

import org.apache.pig.PigServer;

public class PigLatinWordCount {

    public static void main(String[] args) {
        String inputFile = args[0];
        String outputFile = args[1];
        try {
            PigServer pigServer = new PigServer("local");
            pigServer.registerQuery("in_lines = LOAD '" + inputFile + "' AS (line:chararray);");
            pigServer.registerQuery("words = FOREACH in_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;");
            // ...
            // ...
            pigServer.store("ordered_word_count", outputFile);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

More Reading

https://pig.apache.org/docs/r0.14.0/basic.html



Apache Hive:
A mention

Apache Hive

- SQL-style language that compiles into MapReduce jobs in Hadoop

```
DROP TABLE IF EXISTS wiki;
CREATE TABLE wiki (line STRING);
LOAD DATA INPATH 'es-wikipedia-abstracts.txt' OVERWRITE INTO TABLE wiki;
CREATE TABLE wordcount AS
  SELECT word, COUNT(*) AS num
    FROM wiki
    LATERAL VIEW explode(split(text, '\s')) lTable AS word
    GROUP BY word
    ORDER BY num;
```

- Similar to Apache Pig but …
  - Pig more procedural whilst Hive more declarative

Questions?