CC5212-1
PROCESAMIENTO MASIVO DE DATOS
OTOÑO 2019

Lecture 4
Apache Pig

Aidan Hogan
aidhog@gmail.com
HADOOP: WRAPPING UP
**Hadoop: Supermarket Example**

<table>
<thead>
<tr>
<th>ReceiptItems</th>
<th>ReceiptTimes</th>
<th>ItemDetails</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECEIPT ID</td>
<td>ITEM ID</td>
<td>RECEIPT ID</td>
</tr>
<tr>
<td>R1401</td>
<td>I306</td>
<td>R1403</td>
</tr>
<tr>
<td></td>
<td>I306</td>
<td>R1401</td>
</tr>
<tr>
<td>R1402</td>
<td>I007</td>
<td>R1402</td>
</tr>
<tr>
<td>R1403</td>
<td>I306</td>
<td></td>
</tr>
<tr>
<td>R1403</td>
<td>I504</td>
<td></td>
</tr>
</tbody>
</table>

**Compute total sales per hour of the day?**

<table>
<thead>
<tr>
<th>Output</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HOUR</td>
<td>TOTAL</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>18:00–18:59</td>
<td>$2400</td>
</tr>
<tr>
<td>19:00–19:59</td>
<td>$3600</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>
More in Hadoop: Multiple Inputs

```java
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

```java
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);
}
```
More in Hadoop: Number of Reducers

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

<table>
<thead>
<tr>
<th>ReceiptItems</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RECEIPT ID</td>
<td>ITEM ID</td>
</tr>
<tr>
<td>R1401</td>
<td>I306</td>
</tr>
<tr>
<td>R1401</td>
<td>I306</td>
</tr>
<tr>
<td>R1401</td>
<td>I504</td>
</tr>
<tr>
<td>R1402</td>
<td>I007</td>
</tr>
<tr>
<td>R1402</td>
<td>I306</td>
</tr>
<tr>
<td>R1403</td>
<td>I306</td>
</tr>
<tr>
<td>R1403</td>
<td>I504</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ReceiptTimes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RECEIPT ID</td>
<td>TIME</td>
</tr>
<tr>
<td>R1403</td>
<td>19:00</td>
</tr>
<tr>
<td>R1401</td>
<td>18:59</td>
</tr>
<tr>
<td>R1402</td>
<td>19:01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ItemDetails</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEM ID</td>
<td>NAME</td>
</tr>
<tr>
<td>I306</td>
<td>Zanahoria 500g</td>
</tr>
<tr>
<td>I504</td>
<td>CocaCola 3L</td>
</tr>
<tr>
<td>I007</td>
<td>Comfort</td>
</tr>
</tbody>
</table>
| ... | ... | ...

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

<table>
<thead>
<tr>
<th>Output</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HOUR</td>
<td>TOTAL</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>18:00–18:59</td>
<td>$1400</td>
</tr>
<tr>
<td>19:00–19:59</td>
<td>$2600</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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
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 VS. SQL
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);
        FileOutputStream.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);
        FileOutputStream.setOutputPath(job2, new Path(otherArgs[4]));

        job2.setReducerClass(TimesPricesReducer.class);
        job2.setMapOutputKeyClass(LongWritable.class);
        job2.setMapOutputValueClass(Text.class);
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 **User Defined Functions**: call a Java function (or Python, etc.)
- Based on **Pig Relations**

**pig Latin**

/ˈpigˌlātən/

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 ‘pigˌlatn; so chicken soup would be translated to ickenchay ouspay). Pig Latin is typically spoken playfully, as if to convey secrecy.

Translations, word origin, and more definitions
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';
APACHE PIG: AN EXAMPLE
### Pig: Products by Hour

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>Product</th>
<th>Date/Time</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
<tr>
<td>customer413</td>
<td>El_Mercurio</td>
<td>2014-03-31T08:48:03Z</td>
<td>$500</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_Mach3</td>
<td>2014-03-31T08:48:03Z</td>
<td>$8.250</td>
</tr>
<tr>
<td>customer413</td>
<td>Santo_Domingo</td>
<td>2014-03-31T08:48:03Z</td>
<td>$2.450</td>
</tr>
<tr>
<td>customer413</td>
<td>Nescafe</td>
<td>2014-03-31T08:48:03Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer414</td>
<td>Rosas</td>
<td>2014-03-31T08:48:24Z</td>
<td>$7.000</td>
</tr>
<tr>
<td>customer414</td>
<td>300g_Frutillas</td>
<td>2014-03-31T08:48:24Z</td>
<td>$1.230</td>
</tr>
<tr>
<td>customer415</td>
<td>Nescafe</td>
<td>2014-03-31T08:48:35Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer415</td>
<td>12 Huevos</td>
<td>2014-03-31T08:48:35Z</td>
<td>$2.200</td>
</tr>
</tbody>
</table>

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 process 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) ...

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
</tbody>
</table>

...
**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)

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_Mach3</td>
<td>2014-03-31T08:48:03Z</td>
<td>$8.250</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pig: Products by Hour

```pig
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:

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_Mach3</td>
<td>2014-03-31T08:48:03Z</td>
<td>$8.250</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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;
```

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>hour</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>08</td>
<td>$1.240</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_Mach3</td>
<td>08</td>
<td>$8.250</td>
</tr>
</tbody>
</table>

...
Pig: Products by Hour

```pig
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;
```

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>hour</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>08</td>
<td>$1.240</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_Mach3</td>
<td>08</td>
<td>$8.250</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

hourly:
Pig: Products by Hour

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('	') 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);
```

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>hour</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>08</td>
<td>$1.240</td>
</tr>
<tr>
<td>customer413</td>
<td>Gillette_Mach3</td>
<td>08</td>
<td>$8.250</td>
</tr>
<tr>
<td>customer413</td>
<td>Santo_Domingo</td>
<td>08</td>
<td>$2.450</td>
</tr>
</tbody>
</table>
### 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);

<table>
<thead>
<tr>
<th>[item,hour]</th>
<th>cust</th>
<th>item</th>
<th>hour</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>customer412</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td></td>
<td>customer413</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td></td>
<td>customer415</td>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>08</td>
<td>$1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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;
```

<table>
<thead>
<tr>
<th>item</th>
<th>hour</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nescafe</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>[Nescafe,08]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cust</td>
<td></td>
<td></td>
</tr>
<tr>
<td>customer412</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer415</td>
<td>08</td>
<td>$2.000</td>
</tr>
<tr>
<td>400g_Zanahoria</td>
<td>08</td>
<td>$1.240</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cust</td>
<td></td>
<td></td>
</tr>
<tr>
<td>customer413</td>
<td>08</td>
<td>$1.240</td>
</tr>
</tbody>
</table>

...
Pig: Products by Hour

```pig
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;
```

<table>
<thead>
<tr>
<th>item, hour</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

hrItemCnt:
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:

<table>
<thead>
<tr>
<th>[item, hour]</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pig: Products by Hour

```pig
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;
```

<table>
<thead>
<tr>
<th>[item, hour]</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

hrItemCntSorted:
Pig: Products by Hour

```pig
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:**

<table>
<thead>
<tr>
<th>[item, hour]</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Nescafe,08]</td>
<td>3</td>
</tr>
<tr>
<td>[400g_Zanahoria,08]</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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);
  ```

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pig Fields

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

More readable!

Starts at zero.

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>$900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>$2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>$1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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)

\[
\begin{align*}
A &= \text{LOAD} \ 'data' \ \text{AS} \ \text{(cust, item, hour, price)}; \\
B &= \text{FOREACH} \ A \ \text{GENERATE} \ \text{hour + 4 \% 24;} \\
C &= \text{FOREACH} \ A \ \text{GENERATE} \ \text{hour + 4f \% 24;} \\
\end{align*}
\]
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;

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th></th>
<th>t2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t1a</td>
<td>t1b</td>
<td>t1c</td>
<td>t2a</td>
<td>t2b</td>
<td>t2c</td>
</tr>
<tr>
<td>A:</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th></th>
<th>$0</th>
<th>$1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X:</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>
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;
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)})

<table>
<thead>
<tr>
<th>group (c1)</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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)
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';
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);

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th></th>
<th>t2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>t1a</td>
<td>t1b</td>
<td>t1c</td>
<td>t2a</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th></th>
<th>t1a</th>
<th>t1b</th>
<th>t1c</th>
<th>t2a</th>
<th>t2b</th>
<th>t2c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>
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);

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th></th>
<th></th>
<th>t2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t1a</td>
<td>t1b</td>
<td>t1c</td>
<td>t2a</td>
<td>t2b</td>
<td>t2c</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th></th>
<th>t2a</th>
<th>t2b</th>
<th>t2c</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1a</td>
<td>3</td>
<td>t1b</td>
<td>8</td>
<td>t1c</td>
<td>9</td>
</tr>
<tr>
<td>t2a</td>
<td>4</td>
<td>t2b</td>
<td>5</td>
<td>t2c</td>
<td>6</td>
</tr>
<tr>
<td>t2a</td>
<td>1</td>
<td>t2b</td>
<td>4</td>
<td>t2c</td>
<td>7</td>
</tr>
<tr>
<td>t2a</td>
<td>2</td>
<td>t2b</td>
<td>5</td>
<td>t2c</td>
<td>8</td>
</tr>
</tbody>
</table>

Y:
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);

<table>
<thead>
<tr>
<th>group (c1)</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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)
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);
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)

<table>
<thead>
<tr>
<th>group</th>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
APACHE PIG: OPERATORS
Pig Atomic Operators

- **Comparison**
  - $\text{==}, \text{!=}, >, <, \geq, \leq, \text{matches (regex)}$

- **Arithmetic**
  - $+,-,*,/$

- **Reference**
  - $\text{tuple.field, map#value}$

- **Boolean**
  - $\text{AND, OR, NOT}$

- **Casting**
Pig Conditionals

- **Ternary operator:**

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

- **Cases:**

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

• **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

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

<table>
<thead>
<tr>
<th>prod</th>
<th>hour</th>
<th>count</th>
<th>price</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nescafe</td>
<td>08</td>
<td>120</td>
<td>2000</td>
<td>Nescafe</td>
</tr>
<tr>
<td>Nescafe</td>
<td>09</td>
<td>153</td>
<td>2000</td>
<td>Nescafe</td>
</tr>
<tr>
<td>El_Mercurio</td>
<td>08</td>
<td>142</td>
<td>500</td>
<td>El_Mercurio</td>
</tr>
</tbody>
</table>
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:
  – \texttt{PARTITION BY} specifies a UDF for partitioning
  – \texttt{PARALLEL} specifies number of reducers

\[ X = \texttt{JOIN A BY prod, B BY name PARTITION BY org.udp.Partitioner PARALLEL 5;} \]

\[ X = \texttt{GROUP A BY hour PARTITION BY org.udp.Partitioner PARALLEL 5;} \]
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

which 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:**

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**cheap:**

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**premium:**

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pig: Rank

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

\[
\text{numeric:}
\]

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1.240</td>
</tr>
</tbody>
</table>

\[
\text{ranked:}
\]

<table>
<thead>
<tr>
<th>rank</th>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>customer412</td>
<td>1L_Leche</td>
<td>2014-03-31T08:47:57Z</td>
<td>900</td>
</tr>
<tr>
<td>2</td>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>3</td>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>4</td>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pig: Rank

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

**numeric:**

<table>
<thead>
<tr>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer412</td>
<td>1L_Lech</td>
<td>2014-03-31T08:47:57Z</td>
<td>900</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1.240</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**ranked:**

<table>
<thead>
<tr>
<th>rank</th>
<th>cust</th>
<th>item</th>
<th>time</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>customer412</td>
<td>1L_Lech</td>
<td>2014-03-31T08:47:57Z</td>
<td>900</td>
</tr>
<tr>
<td>2</td>
<td>customer413</td>
<td>400g_Zanahoria</td>
<td>2014-03-31T08:48:03Z</td>
<td>1.240</td>
</tr>
<tr>
<td>3</td>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>3</td>
<td>customer412</td>
<td>Nescafe</td>
<td>2014-03-31T08:47:57Z</td>
<td>2.000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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

```pig
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

```pig
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

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 "\s";

-- 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

```sql
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?