Big Data

Diplomado de Datos 2021

Clase 3: Pig

Aidan Hogan
aidhog@gmail.com

HADOOP VS. SQL

Hadoop: (ʘ_ʘ)

```
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 != 5) {
            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);
```

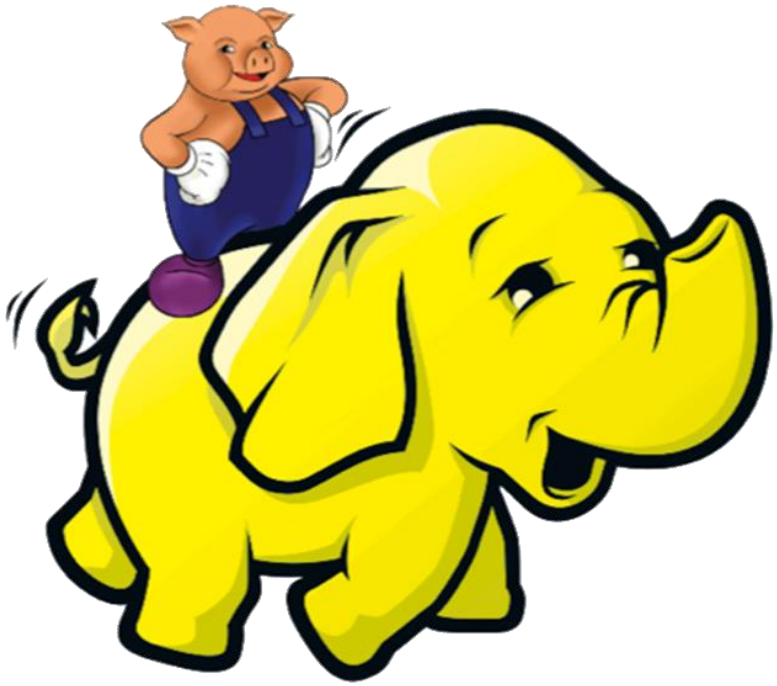
¿Por qué no usamos SQL entonces?

Los sistemas de bases de datos relacionales no están optimizados, en general, para procesar datos a gran escala, sino están optimizados para contestar consultas que devuelvan pocos resultados rapidamente.

Pero la pregunta fue: ¿Por qué no usamos SQL?
No fue: ¿Por qué no usamos un sistema de bases de datos relacionales?

Apache Pig:
Un paneo general

Apache Pig



- Crea tareas de **Hadoop**
- Usa un lenguaje de “scripting” de alto nivel llamado **Pig Latin**
- Puede integrar **funciones definidas por el usuario**: llamar a una función en Java (o Python, Ruby, etc.)
- Basado en **Relaciones de Pig**

Apache Pig:
Un ejemplo

Pig: Productos por hora

transact.txt

```
customer412     1L_Leche          2014-03-31T08:47:57Z     900
customer412     Nescafe           2014-03-31T08:47:57Z     2000
customer412     Nescafe           2014-03-31T08:47:57Z     2000
customer413     400g_Zanahoria    2014-03-31T08:48:03Z     1240
customer413     El_Mercurio       2014-03-31T08:48:03Z     500
customer413     Gillette_Mach3    2014-03-31T08:48:03Z     8250
customer413     Santo_Domingo     2014-03-31T08:48:03Z     2450
customer413     Nescafe           2014-03-31T08:48:03Z     2000
customer414     Rosas             2014-03-31T08:48:24Z     7000
customer414     Chocolates        2014-03-31T08:48:24Z     9230
customer414     300g_Frutillas    2014-03-31T08:48:24Z     1230
customer415     Nescafe           2014-03-31T08:48:35Z     2000
customer415     12 Huevos         2014-03-31T08:48:35Z     2200
…
```

Para cada producto con un precio mayor que $1.000, encontrar el número de unidades vendidas por hora (en transacciones distintas)

Pig: Productos por hora

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

Funciones definidas por el usuario, escritas en Java (o Python, Ruby, etc. ...)
User Defined Function = UDF

```
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: Productos por hora

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

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Pig: Productos por hora

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

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
```

premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
customer413	Gillette_Mach3	2014-03-31T08:48:03Z	8250
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
```

premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
customer413	Gillette_Mach3	2014-03-31T08:48:03Z	8250
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
```

hourly:

cust	item	hour	price
customer412	Nescafe	08	2000
customer412	Nescafe	08	2000
customer413	400g_Zanahoria	08	1240
customer413	Gillette_Mach3	08	8250
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
```

hourly:

cust	item	hour	price
customer412	Nescafe	08	2000
customer412	Nescafe	08	2000
customer413	400g_Zanahoria	08	1240
customer413	Gillette_Mach3	08	8250
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
```

unique:

cust	item	hour	price
customer412	Nescafe	08	2000
customer413	400g_Zanahoria	08	1240
customer413	Gillette_Mach3	08	8250
customer413	Santo_Domingo	08	2450
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD 'transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
grunt> hourly = FOREACH premium GENERATE cust, item, org.udf.ExtractHour(time) AS hour, price;
grunt> unique = DISTINCT hourly;
```

unique:

cust	item	hour	price
customer412	Nescafe	08	2000
customer413	400g_Zanahoria	08	1240
customer413	Gillette_Mach3	08	8250
customer413	Santo_Domingo	08	2450
…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
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	2000
	customer413	Nescafe	08	2000
	customer415	Nescafe	08	2000
[400g_Zanahoria,08]	customer413	400g_Zanahoria	08	1240
…	…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
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	2000
	customer413	Nescafe	08	2000
	customer415	Nescafe	08	2000
[400g_Zanahoria,08]	customer413	400g_Zanahoria	08	1240
…	…	…	…	…

Pig: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
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: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
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: Productos por hora

```
grunt> REGISTER userDefinedFunctions.jar;
grunt> raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
grunt> premium = FILTER raw BY price >= 1000;
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: Esquema

Relaciones en Pig

- Relaciones en Pig: Como tablas relacionales
  - En este caso las tuplas pueden ser incompletas
  - Las relaciones están desordenadas por defecto

- Esquema en Pig: Nombres para campos, 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	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Campos en Pig

- Campos en Pig:
  - Se referencian por nombre
    - `premium = FILTER raw BY price >= 1000;`
  - … o posición
    - `premium = FILTER raw BY $3 >= 1000;`

¡Más legible!

Empieza con cero

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Apache Pig: Tipos

Tipos simples en Pig

- Tipos en Pig:
  - ```
    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
    ```

Tipos en Pig: "Duck Typing" (tipos flexibles)

- ¿Qué pasa si se omiten los tipos?
  - Por defecto los campos son `bytearray`
  - Conversiones implícitas contextuales (~"duck typing")

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

hour un integer

hour un float

Tipos complejos en Pig

- `tuple`: Una fila en una tabla / una lista de campos
  - ej., `(customer, Nescafe, 08, 2000)`
  - ej., `(customer, (Nescafe, 08, 2000))`

- `bag` (saco): Un multiconjunto de tuplas (permite duplicados)
  - ej., `{ (cust412, 08, 2000), (cust412, 08, 2000), (cust413, 08, 8250) }`

Tipos complejos en Pig: Tuplas

```
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

Tipos complejos en Pig: Tuplas

```
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

Tipos complejos en Pig: Sacos

```
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

Tipos complejos en Pig: Sacos

```
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

Apache Pig:
Desanidar ("flatten")

Pig: Desanidar tuplas

```
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: Desanidar tuplas

```
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: Desanidar tuplas

```
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: Desanidar tuplas

```
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: Desanidar sacos

```
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: Desanidar sacos

```
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: Desanidar sacos

```
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: Desanidar sacos

```
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:
Operadores

Operadores atómicos en Pig

- Comparación
  `==`, `!=`, `>`, `<`, `>=`, `<=`, `matches` (regex)
- Aritmética
  `+`, `-`, `*`, `/`
- Referencia
  `tuple.field`
- Booleanos
  `AND`, `OR`, `NOT`
- Conversión ("Casting")

Condicionales en Pig

- Operador ternario:

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

- Casos:

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

Agregación en Pig

Se puede usar GROUP con múltiples relaciones o COGROUP con una única (pero COGROUP se considera más legible para múltiples relaciones)

- Agrupamiento:
  - GROUP: agrupamiento con una sola relación
    - `GROUP premium BY (item, hour);`
  - COGROUP: agrupamiento con múltiples relaciones
    - `COGROUP premium BY (item, hour), cheap BY (item, hour);`
- Operadores de agregación:
  - AVG, MIN, MAX, SUM, COUNT, SIZE, CONCAT

Joins en Pig

```
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

Joins en Pig

- Inner join: El ejemplo anterior (el join por defecto)
- Self join: Genera un alias y hace join con él
- Outer joins: `LEFT` / `RIGHT` / `FULL`
- Producto cruz: `CROSS`

¿Saben (o recuerdan) la diferencia entre un `INNER JOIN`, `OUTER JOIN` / `LEFT` / `RIGHT` / `FULL` y un `CROSS PRODUCT`?

Implementación de agregación / join en Pig

- Partición configurable / número de reductores:
  - `PARTITION BY` especifica una UDF para particionar
  - `PARALLEL` especifica el número de reductores

```
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: Desambiguación

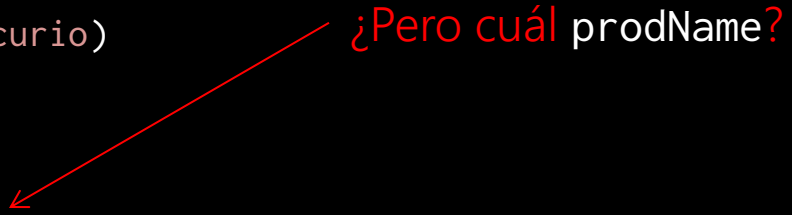
```
cat data1;
(Nescafe,08,120)
(El_Mercurio,08,142)
(Nescafe,09,153)

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: Filtrar

```
raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
premium = FILTER raw BY price>=1000;
```

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

premium:

cust	item	time	price
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Pig: Repartición ("Split")

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

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

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	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Pig: Unión

```
cheap = LOAD ‘cheap.txt' USING PigStorage('\t') AS (cust, item, time, price);
premium = LOAD ‘prem.txt' USING PigStorage('\t') AS (cust, item, time, price);
all = UNION cheap, premium;
```

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	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

all:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Pig: Ordenar

```
raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
ordered = ORDER raw BY price ASC, cust DESC;
```

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

ordered:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
…	…	…	…

Pig: Rank

```
raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
ranked = RANK raw;
```

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

ranked:

rank	cust	item	time	price
1	customer412	1L_Leche	2014-03-31T08:47:57Z	900
2	customer412	Nescafe	2014-03-31T08:47:57Z	2000
3	customer412	Nescafe	2014-03-31T08:47:57Z	2000
4	customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
...	…	…	…	…

Pig: Rank

```
raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
ranked = RANK raw BY price ASC, cust DESC;
```

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

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	1240
3	customer412	Nescafe	2014-03-31T08:47:57Z	2000
3	customer412	Nescafe	2014-03-31T08:47:57Z	2000
...	…	…	…	…

Pig: Limitar

```
raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
firsttwo = LIMIT raw 2;
```

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

firsttwo:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000

Pig: Muestra

```
raw = LOAD ‘transact.txt' USING PigStorage('\t') AS (cust, item, time, price);
sample = SAMPLE raw 0.5;
```

raw:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer412	Nescafe	2014-03-31T08:47:57Z	2000
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

sample:

cust	item	time	price
customer412	1L_Leche	2014-03-31T08:47:57Z	900
customer413	400g_Zanahoria	2014-03-31T08:48:03Z	1240
…	…	…	…

Pig: Otros operadores

- `NATIVE`: Ejecutar un `.jar` nativo de Hadoop

Apache Pig: Nulos

Pig: Nulos

- Nulos representan información que falta

```
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: Nulos con Joins

- Nulos representan información que falta
- Su comportamiento cumple con las defs. de SQL

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

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

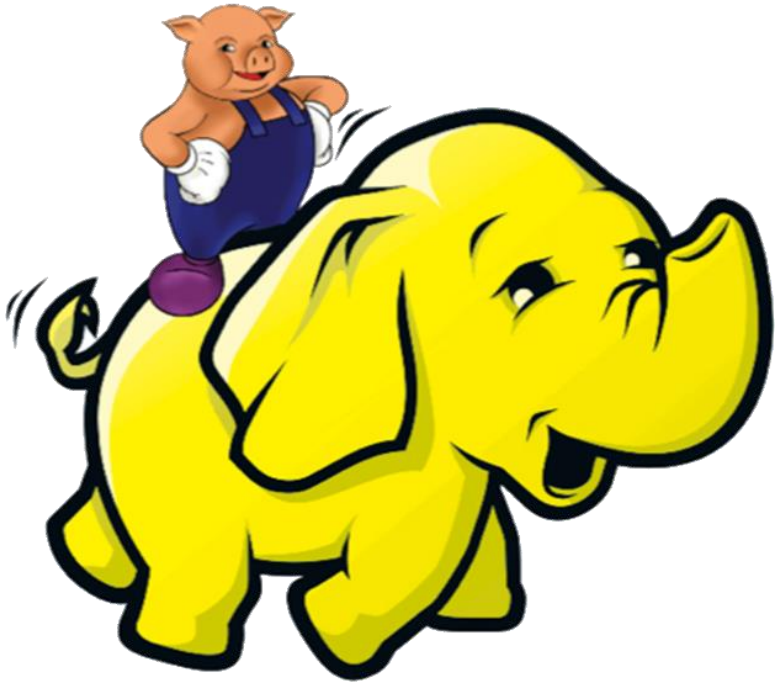
A = LOAD 'data1' AS (prodName:charArray, hour:int, count:int);
B = LOAD 'data2' AS (price:int, prodName:charArray);
X = JOIN A BY prodName, B BY prodName;

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

Apache Pig: Ejecución

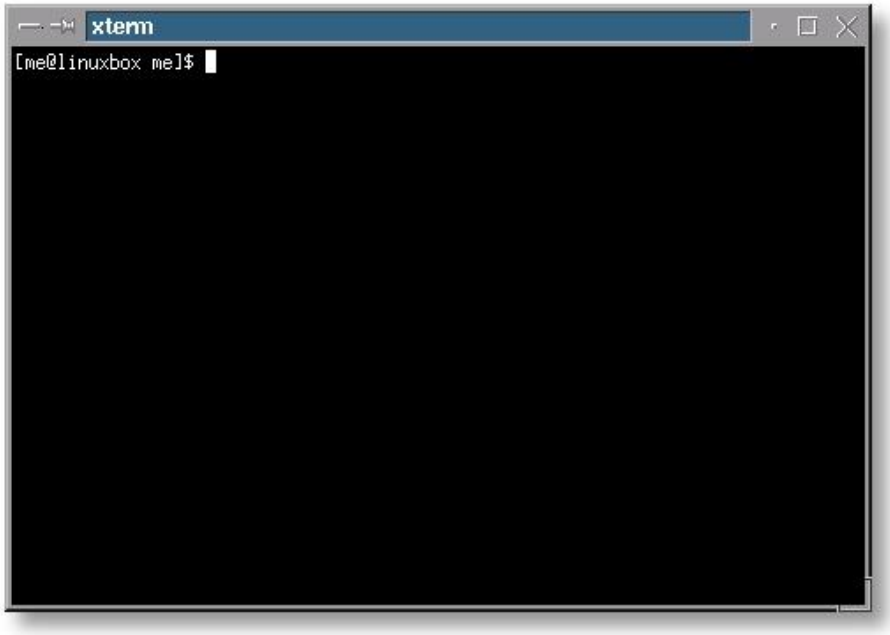
Traducido a MapReduce en Hadoop

- Pig es sólo una interfaz / un lenguaje de scripting para MapReduce



Ejecutar 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';
```



Ejecutar 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';
```

Ejecutar Pig: (iii) Desde código

```
package scratch;

import org.apache.pig.PigServer;

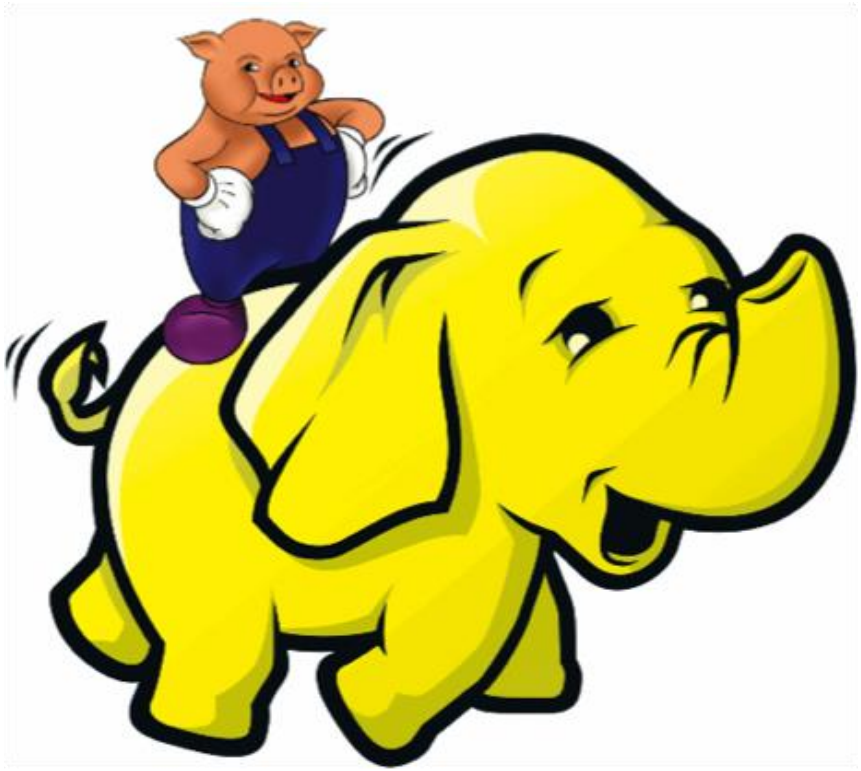
public class PigLatinWordCount {

    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();
        }
    }

}
```

Si quieren leer más

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



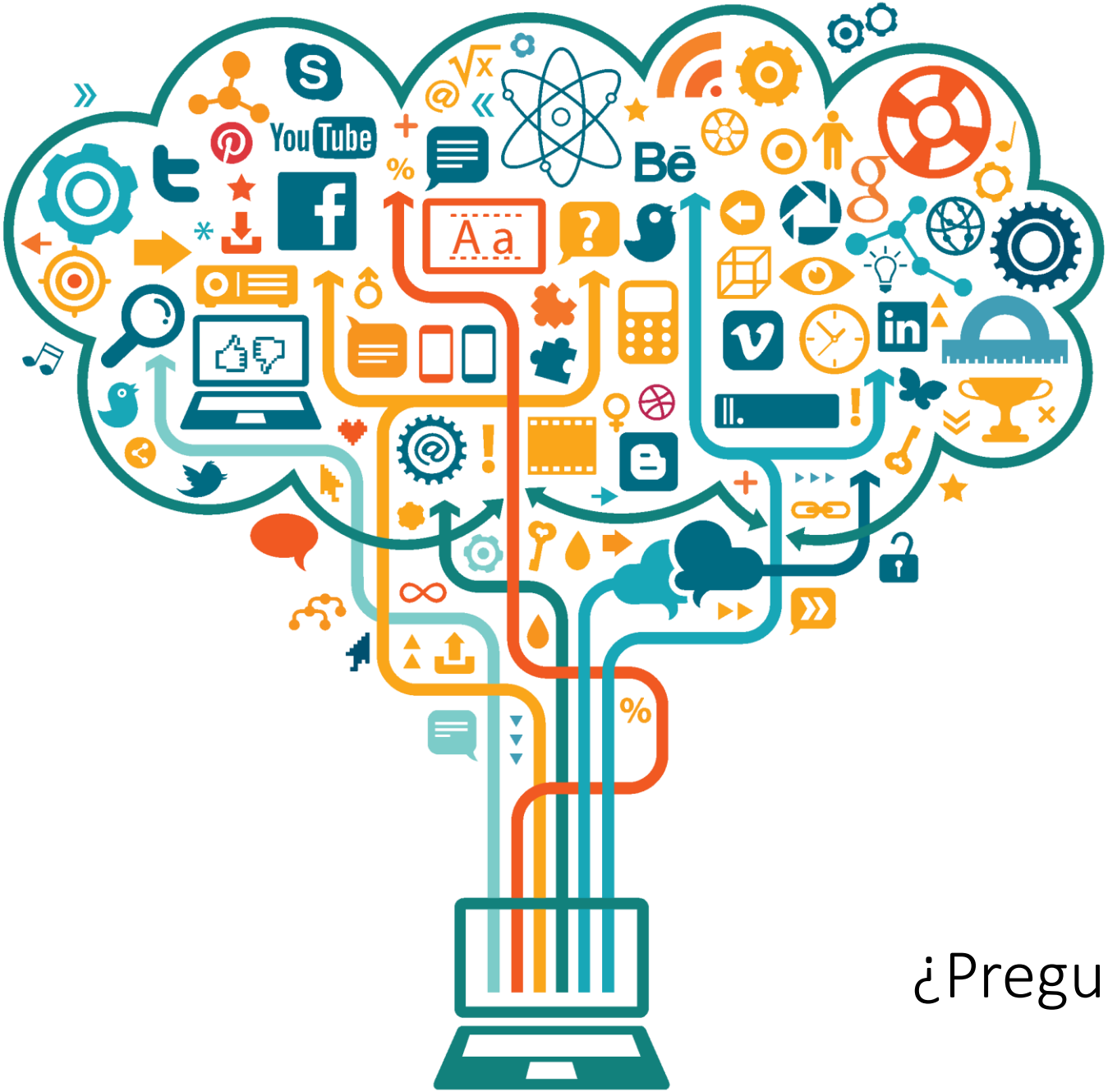
Apache Hive:
Una mención ...

Apache Hive

- HiveQL: Lenguaje al estilo SQL que se compila en tareas de MapReduce en 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;
```

- Parecido a Apache Pig pero …
  - Pig es más procedural y Hive es más declarativo
  - Pig permite desarrollo más interactivo



¿Preguntas?