

Hadoop: Hive y Pig

*Máster en Business Intelligence e
Innovación Tecnológica*

Introducción

- Solución de DWH en el stack de Hadoop.
- Proporciona un lenguaje SQL-like llamado HiveQL:
 - Minimiza la curva de aprendizaje
 - Audiencia Target: Analistas de datos.
- Se inicia el desarrollo por Facebook en 2007
- Actualmente es un proyecto de Apache bajo la marca Hadoop.
 - <http://hive.apache.org>



Hive proporciona

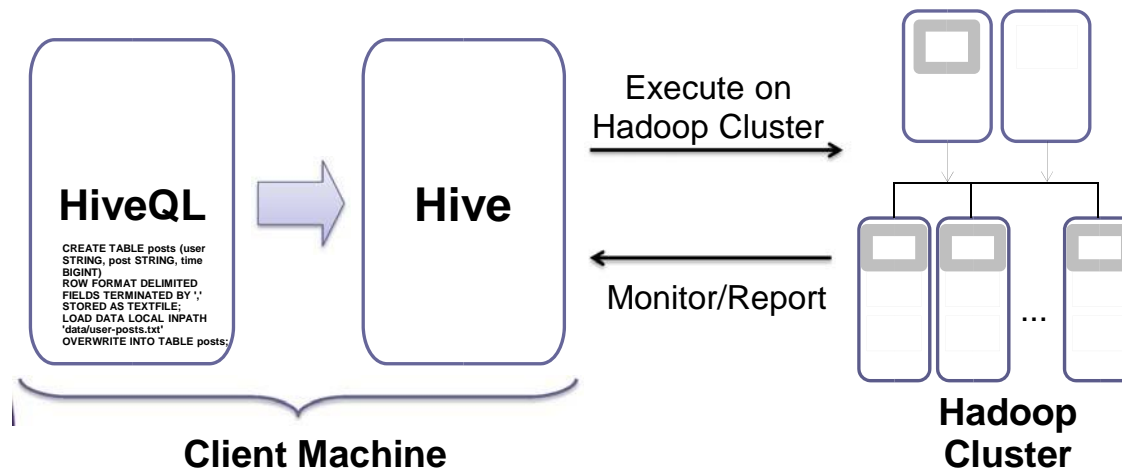
- Habilita la prestación de una estructura para varios formatos de datos
- Una interfaz simple para el desarrollo de queries ad-hoc, analizar y resumir grandes cantidades de datos.
- Acceso a varios almacenes de datos como HBase o HDFS.
- Se puede acceder vía consola, algún front-end, o JDBC. Esto último facilita su integración en aplicaciones.

Hive NO proporciona

- Queries de baja latencia o en tiempo real
- Incluso consultar bajos volúmenes de datos puede costar varios minutos.
- Diseñado para la escalabilidad y facilidad de uso en lugar de las respuestas de baja latencia.

Hive

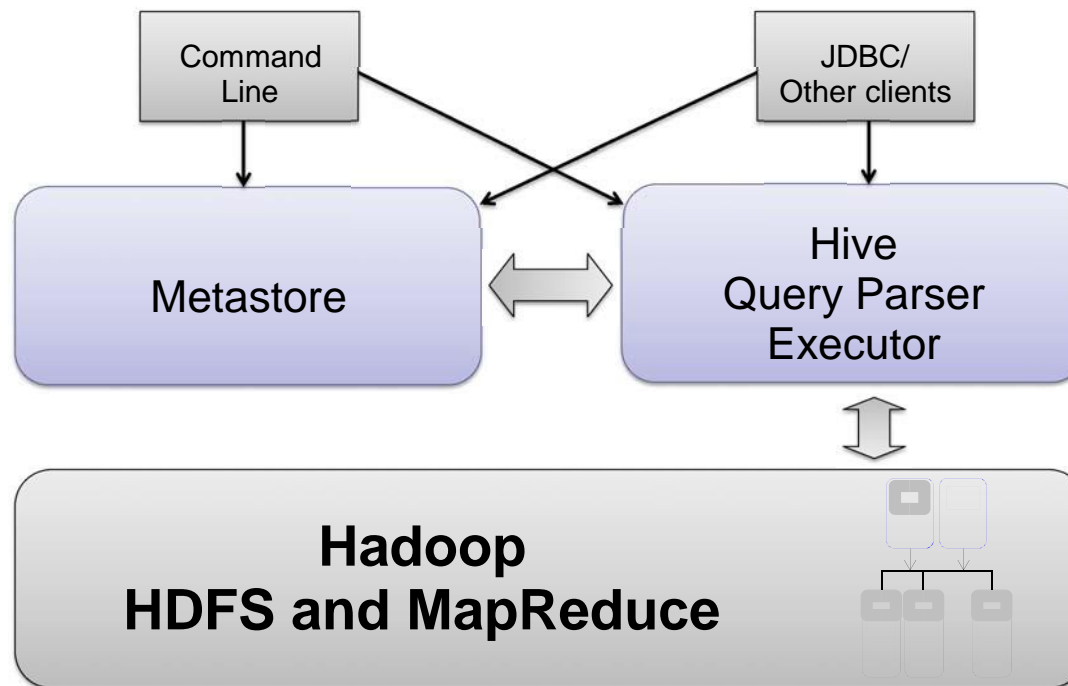
- Traduce sentencias HiveQL en conjuntos de Jobs MapReduce que se ejecutan en un Cluster Hadoop.



Hive Metastore

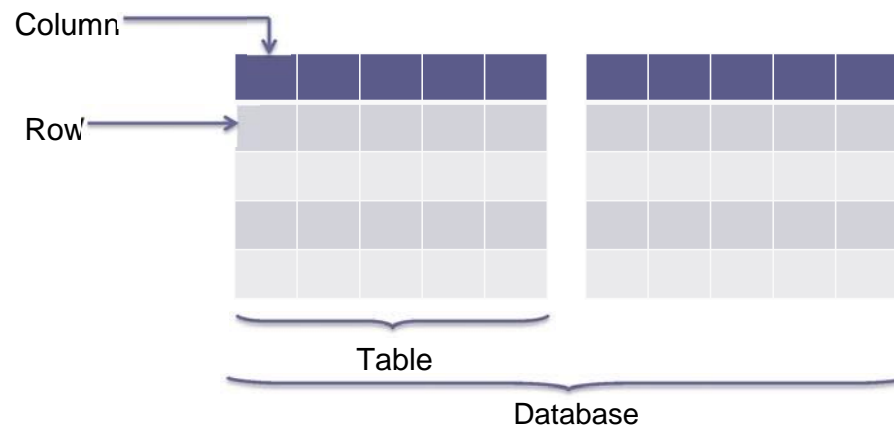
- Para soportar características como esquemas o partición de datos, Hive almacena su Metadata en una BDD relacional.
- Está empaquetada con Derby, una BDD SQL ligera embebida.
 - La implementación por defecto es buena para evaluación y testeo.
 - El esquema no está compartido entre usuarios, por lo que cada usuario tiene su propia instancia.
 - Almacenada en el directorio metastore_db, reside en el directorio de arranque de Hive.
- Se puede cambiar fácilmente a otro SGBD como MySQL.

Arquitectura Hive



Conceptos de Hive

Reutilizados de BDD Relacionales: Base de datos, tabla, fila, columna.



Ejemplos: Create table

```
$ cd $PLAY AREA
```

Lanza la interfaz de Hive

```
$ hive
```

Ubicación logs

```
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201208022144_2014345460.txt
```

```
hive> !cat data/user-posts.txt;
```

Permite ejecutar sentencias en local

```
user1,Funny Story,1343182026191
```

```
user2,Cool Deal,1343182133839
```

```
user4,Interesting Post,1343182154633
```

```
user5,Yet Another Blog,13431839394
```

```
hive>
```

Los valores se muestran separados por ",", cada fila representa un registro

Ejemplos: Create table

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
OK
Time taken: 10.606 seconds
```

Crea una tabla de tres columnas
Determina como se procesa
el fichero y como se almacenará

```
hive> show tables;
OK
posts
Time taken: 0.221 seconds
```

→ Muestra las tablas de una base de datos

```
hive> describe posts;
OK
user      string
post      string
time      bigint
Time taken: 0.212 seconds
```

→ Describe la estructura de posts

Ejemplos: Cargar datos en una tabla

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'  
> OVERWRITE INTO TABLE posts;
```

Copying data from file:/home/hadoop/Training/play_area/data/user-posts.txt

Copying file: file:/home/hadoop/Training/play_area/data/user-posts.txt

Loading data to table default.posts

Deleted /user/hive/warehouse/posts

OK

Time taken: 5.818 seconds

hive>

Los datos existentes en posts se eliminan, la información de user-posts se almacena en la tabla

```
$ hdfs dfs -cat /user/hive/warehouse/posts/user-posts.txt
```

user1,Funny Story,1343182026191



user2,Cool Deal,1343182133839


user4,Interesting Post,1343182154633

user5,Yet Another Blog,13431839394

Si no se especifica lo contrario, Hive almacena su contenido dentro de la carpeta warehouse

Ejemplos: Consultar datos en una tabla

hive> **select count (1) from posts;**  Contar el número de registros en posts
Total MapReduce jobs = 1  La sentencia HQL se transforma en un job MapReduce
Launching Job 1 out of 1

....
Starting Job = job_1343957512459_0004, Tracking URL =
http://localhost:8088/proxy/application_1343957512459_0004/
Kill Command = hadoop job -Dmapred.job.tracker=localhost:10040 -kill
job_1343957512459_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2012-08-02 22:37:24,962 Stage-1 map = 0%, reduce = 0%
2012-08-02 22:37:30,497 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-08-02 22:37:31,577 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-08-02 22:37:32,664 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.64 sec
MapReduce Total cumulative CPU time: 2 seconds 640 msec
Ended Job = job_1343957512459_0004
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.64 sec HDFS Read: 0 HDFS Write: 0
SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 640 msec
OK
4  El resultado son 4 registros
Time taken: 14.204 seconds

Ejemplos: Consultar datos en una tabla

```
hive> select * from posts where user="user2";
```

→ Seleccionar posts de "user2"

```
...  
...  
OK  
user2 Cool Deal      1343182133839  
Time taken: 12.184 seconds
```


```
hive> select * from posts where time<=1343182133839 limit 2; -
```

↗ Seleccionar posts cuyo timestamp es menor o igual al especificado

```
...  
...  
OK  
user1 Funny Story    1343182026191  
user2 Cool Deal      1343182133839  
Time taken: 12.003 seconds  
hive>
```

↘ Generalmente hay muchos resultados en un BigData, se utiliza la función limit

Ejemplos: Borrar datos de una tabla

```
hive> DROP TABLE posts;  Borrar una tabla  
OK  
Time taken: 2.182 seconds
```

```
hive> exit;
```

```
$ hdfs dfs -ls /user/hive/warehouse/  
$
```

 Si no existen tablas el resultado está vacío

Cargando datos

- Utilizando una ubicación dentro de HDFS.

```
hive> CREATE EXTERNAL TABLE posts  
      > (user STRING, post STRING, time BIGINT)  
      > ROW FORMAT DELIMITED  
      > FIELDS TERMINATED BY ','  
      > STORED AS TEXTFILE  
      > LOCATION '/training/hive/';
```

OK

Time taken: 0.077 seconds

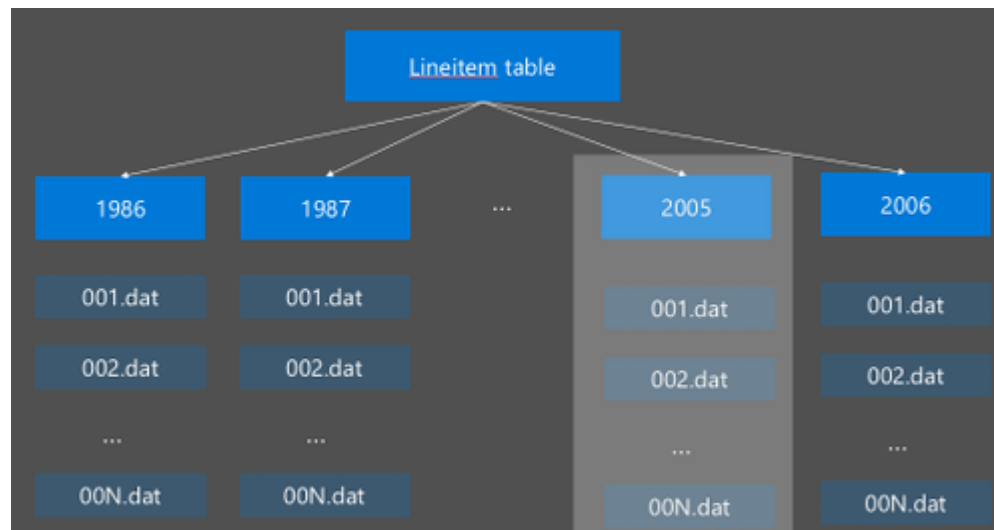
hive>



Hive cargará todos los ficheros que se encuentren bajo la ruta especificada

Particiones de datos

- Para incrementar el rendimiento, Hive tiene la capacidad de particionar datos.
 - Los valores de una columna particionada se dividen en segmentos.
 - En tiempo de consulta se pueden ignorar particiones enteras.
 - Son similares a los índices de las BDD relacionales pero no tan granulares.
- Se deben definir por los usuarios. Al insertar datos se especifica la partición.
- En tiempo de consulta, si es apropiado, Hive puede filtrar las particiones.



Crear una tabla particionada

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
> PARTITIONED BY(country STRING)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
```

OK

Time taken: 0.116 seconds

```
hive> describe posts;
```

OK

user string

post string

time bigint

countrystring

Time taken: 0.111 seconds

Dentro del esquema no vemos diferencias entre las columnas de partición y las columnas de datos

```
hive> show partitions posts;
```

OK


Time taken: 0.102 seconds

```
hive>
```

Tabla de partición basada en el valor de un país.

Cargar datos en una tabla particionada


```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'
> OVERWRITE INTO TABLE posts;
FAILED: Error in semantic analysis: Need to specify partition
columns because the destination table is partitioned
```



Desde que se ha definido Posts para ser particionada, cada insert debe especificar la particion

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'
> OVERWRITE INTO TABLE posts PARTITION(country='US');
OK
Time taken: 0.225 seconds
```

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-AUSTRALIA.txt'
> OVERWRITE INTO TABLE posts PARTITION(country='AUSTRALIA');
OK
Time taken: 0.236 seconds
hive>
```



Cada fila se carga en una partición separada: la información se distribuye por país

Almacenamiento de particiones

- Las particiones se almacenan en directorios separados:

```
hive> show partitions posts;
```

```
OK
```

```
country=AUSTRALIA
```

```
country=US
```

```
Time taken: 0.095 seconds
```

```
hive> exit;
```

```
$ hdfs dfs -ls -R /user/hive/warehouse/posts
```

```
/user/hive/warehouse/posts/country=AUSTRALIA
```

```
/user/hive/warehouse/posts/country=AUSTRALIA/user-posts-AUSTRALIA.txt
```

```
/user/hive/warehouse/posts/country=US
```

```
/user/hive/warehouse/posts/country=US/user-posts-US.txt
```

Existe un directorio para cada particion



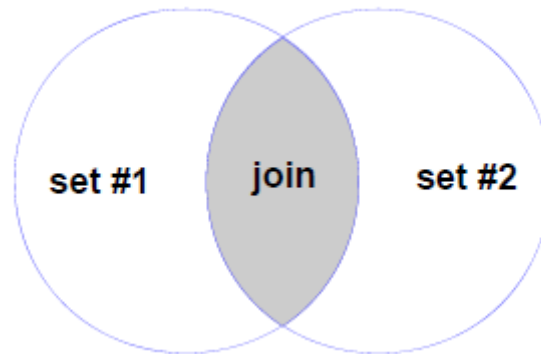
Consultando datos de una tabla particionada

- No hay diferencia en la sintaxis. Si se especifica la columna particionada en la cláusula where, se pueden ignorar ciertas particiones.

```
hive> select * from posts where country='US' limit 10;
OK
user1 Funny Story 1343182026191      US
user2 Cool Deal   1343182133839      US
user2 Great Interesting Note 13431821339485      US
user4 Interesting Post 1343182154633      US
user1 Humor is good 1343182039586      US
user2 Hi I am user #2 1343182133839      US
Time taken: 0.197 seconds
```

Joins

- Los Joins son sencillos si se conoce el concepto en BDD Relacionales.
- Soporta Outer Joins: left, right y full.
- Se pueden unir múltiples tablas
- El join por defecto es inner.
 - Las filas se unen mediante matching de claves.
 - Las filas sin matching no se incluyen en los resultados.



Inner Join Simple

```
hive> select * from posts limit 10;
```

```
OK
```

user1	Funny Story	1343182026191
user2	Cool Deal	1343182133839
user4	Interesting Post	1343182154633
user5	Yet Another Blog	1343183939434

```
Time taken: 0.108 seconds
```

```
hive> select * from likes limit 10;
```

```
OK
```

user1	12	1343182026191
user2	7	1343182139394
user3	0	1343182154633
user4	50	1343182147364

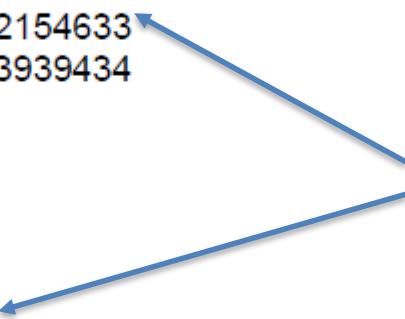
```
Time taken: 0.103 seconds
```

```
hive> CREATE TABLE posts_likes (user STRING, post STRING, likes_count INT);
```

```
OK
```

```
Time taken: 0.06 seconds
```

Queremos unir los dos datasets en una única tabla que contenga usuario, post y contador de likes



Inner Join Simple

```
hive> INSERT OVERWRITE TABLE posts_likes  
      > SELECT p.user, p.post, l.count  
      > FROM posts p JOIN likes l ON (p.user = l.user);
```

OK

Time taken: 17.901 seconds



Se hace un join de las dos tablas en función de la columna user

```
hive> select * from posts_likes limit 10;
```

OK

user1	Funny Story	12
-------	-------------	----

user2	Cool Deal	7
-------	-----------	---

user4	Interesting Post	50
-------	------------------	----

Time taken: 0.082 seconds

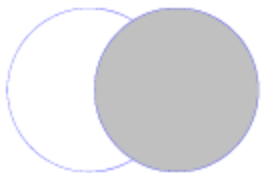
hive>

Outer Join



Left Outer

- Row from the first table are included whether they have a match or not. Columns from the unmatched (second) table are set to null.



Right Outer

- The opposite of Left Outer Join: Rows from the second table are included no matter what. Columns from the unmatched (first) table are set to null.



Full Outer

- Rows from both sides are included. For unmatched rows the columns from the 'other' table are set to null.

Outer Join

```
SELECT p.*, l.*  
FROM posts p LEFT OUTER JOIN likes l ON (p.user = l.user)  
limit 10;
```

```
SELECT p.*, l.*  
FROM posts p RIGHT OUTER JOIN likes l ON (p.user = l.user)  
limit 10;
```

```
SELECT p.*, l.*  
FROM posts p FULL OUTER JOIN likes l ON (p.user = l.user)  
limit 10;
```


Introducción

“es una plataforma para analizar grandes grupos de datos que consiste en un lenguaje de alto nivel para expresar programas de análisis de datos junto con una infraestructura para evaluar estos programas”

- Proyecto de Apache: <http://pig.apache.org>
- Pig es una abstracción de Hadoop:
 - Proporciona un lenguaje de programación de alto nivel para procesar datos.
 - Se convierte en MapReduce y se ejecuta en clusters Hadoop.
- Está ampliamente aceptado y usado: Yahoo!, Twitter, Netflix...



Pig y MapReduce

- MapReduce necesita programadores:
 - Se deben plantear las fases de Map y de Reduce.
 - Necesitará programadores Java.
- Pig proporciona un lenguaje de alto nivel que puede ser usado por analistas, data scientists, estadísticos, etc.
- Originalmente se desarrolló por Yahoo! pensando en permitir a los analistas de datos acceder a la información.
- Características: unir datasets, ordenar datasets, filtrar, tipos de datos, Group By, crear funciones definidas por usuarios,...

Usos: procesos ETL (Extract Transform Load), investigación de altos volúmenes de información (logs, etc.) ,...

Componentes

- Pig Latin:
 - Lenguaje de programación basado en comandos.
 - Diseñado especialmente para la transformación y flujos de expresiones.
- Entorno de ejecución:
 - El entorno dónde Pig Latin es ejecutado.
 - Soporte para ejecución local y en nodos Hadoop.
- El compilador de Pig convierte Pig Latin a Map Reduce:
 - El compilador se esfuerza en optimizar la ejecución.
 - Se obtiene optimización a medida que se aplican actualizaciones.
- Modos de ejecución:
 - Local: en una JVM, útil para desarrollo y experimentación.
 - Hadoop Mode: los Jobs MapReduce se ejecutan en un cluster.

Modo Hadoop

```
- 1: Load text into a bag, where a row is a line of  
text  
lines = LOAD '/training/playArea/hamlet.txt' AS  
(line:chararray);  
- 2: Tokenize the provided text  
tokens = FOREACH lines GENERATE  
flatten(TOKENIZE(line)) AS token:chararray;
```

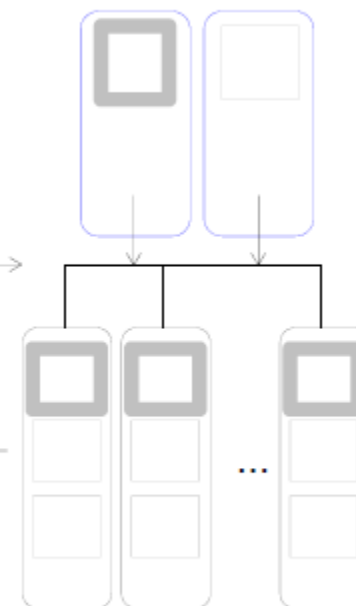
PigLatin.pig

Parse Pig script and
compile into a set of
MapReduce jobs



Execute on
Hadoop Cluster

Monitor/Report



**Hadoop
Cluster**

Ejecutando Pig

- Por Script:
 - Ejecuta comandos desde un fichero.
 - `$pig scriptFile.pig`
- Grunt:
 - Shell interactiva para ejecutar programas.
 - Se pueden ejecutar scripts mediante `run` o `exec`.
- Embebidos:
 - Ejecutan comandos Pig usando la clase `PigServer`
 - Se tiene acceso programático a Grunt desde la clase `PigRunner`.

Conceptos

- Contruyendo bloques:
 - Campo (field): trozo de información
 - Tupla: conjunto ordenado de datos. Representado por “(“ y “)”.
 - Bag: colección de tuplas, representado por “{“ y “}”.
- Similar a bases de datos relacionales:
 - Bag es una tabla en la base de datos.
 - Tupla es una fila en la base de datos.
 - Bag no requiere que las tuplas tengan el mismo número de datos

Ejemplo Pig Latin

```
$ pig
grunt> cat /training/playArea/pig/a.txt
a      1
d      4
c      9
k      6
grunt> records = LOAD '/training/playArea/pig/a.txt' as
(letter:chararray, count:int);
grunt> dump records;
...
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 50% complete
2012-07-14 17:36:22,040 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 100% complete
...
(a,1)
(d,4)
(c,9)
(k,6)
grunt>
```

Arranca Grunt con el modo MapReduce por defecto

Grunt soporta comandos del sistema

Carga contenido de ficheros de texto en conjuntos de registros llamados Bag

Muestra el contenido del Bag records por pantalla

Registros

Dump y Store

- No se efectúan operaciones hasta que se encuentra uno de estos elementos. Pig parsea, valida y analiza las sentencias, pero no las ejecuta.
- Dump: muestra contenido por pantalla.
- Store: almacena datos, típicamente en un fichero.

No se ejecuta nada: Pig optimiza todo ese trozo de script

```
records = LOAD '/training/playArea/pig/a.txt' as  
(letter:chararray, count:int);  
...  
...  
...  
...  
...  
DUMP final_bag;
```

Lo divertido empieza aquí

Grandes cantidades de datos

- En Hadoop la información suele ser muy elevada y no tiene sentido mostrar los datos por pantalla.
- Es más común almacenar estos datos en Hadoop (HDFS, Hbase) con el comando STORE.
- Para debug o información, se pueden imprimir sub-sets de datos.

```
grunt> records = LOAD '/training/playArea/pig/excite-small.log'  
AS (userId:chararray, timestamp:long, query:chararray);  
grunt> toPrint = LIMIT records 5;  
grunt> DUMP toPrint;
```



Sólo se mostrarán 5 registros

Comando Load

```
LOAD 'data' [USING function] [AS schema];
```

- Data: nombre del fichero o directorio.
- Using: especifica la función utilizada para cargar. Por defecto se utiliza PigStorage, que convierte cada línea en campos usando un delimitador (por defecto tab, pero se puede modificar).
- AS: asigna un esquema para la entrada de datos: asigna nombres a los datos y declara tipos.

```
records =  
    LOAD '/training/playArea/pig/excite-small.log'  
    USING PigStorage()  
    AS (userId:chararray, timestamp:long, query:chararray);
```

Schema Data Types

Tipo	Descripcion	Ejemplo
Simples		
int	Entero de 32 bits con signo	10
long	Entero de 64 bits con signo	10L o 10l
float	Número de 32 bits en coma flotante	10.5F o 10.5f
double	Número de 64 bits en coma flotante	10.5 o 10.5 e2 o 10.5 E2
Arrays		
chararray	Array de caracteres (String)	Hola mundo
bytearray	Array de bytes (blob)	
Tipos complejos		
tuple	Un conjunto ordenado de campos	(10,9)
bag	Una colección de tuplas	{{(10,9), (12,8)}}
map	Una colección de tuplas	[open#apache]

Herramientas de diagnóstico

- Mostrar la estructura de un Bag:

```
grunt> DESCRIBE <bag_name>;
```

- Mostrar un plan de ejecución:
 - Realiza varios informes:
 - Plan lógico
 - Plan MapReduce

```
grunt> EXPLAIN <bag_name>;
```

- Mostrar como Pig transforma la información

```
grunt> ILLUSTRATE <bag_name>;
```

Agrupaciones

```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS
(c:chararray);
grunt> describe chars;
chars: {c: chararray}
grunt> dump chars;
(a)
(k)
```

...
...
(k)
(c)
(k)

Crea una nueva bag con un elemento
llamado Group y uno llamado chars

```
grunt> charGroup = GROUP chars by c;
grunt> describe charGroup;
charGroup: {group: chararray, chars: {(c: chararray)}}
grunt> dump charGroup;
(a, {(a), (a), (a)})
(c, {(c), (c)})
(i, {(i), (i), (i)})
(k, {(k), (k), (k), (k)})
(l, {(l), (l)})
```

El bag "chars" se agrupa por c. El
elemento group contendrá claves únicas

Chars a su vez es una Bag y contiene las
tuplas del bag chars cuyo valor coincide
con c

El comando illustrate

```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS (c:chararray);
grunt> charGroup = GROUP chars by c;
grunt> ILLUSTRATE charGroup;
```

chars	c:chararray
	c
	c

charGroup	group:chararray	chars:bag{:tuple(c:chararray)}
	c	{(c), (c)}

Inner y Outer Bags

```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS (c:chararray);  
grunt> charGroup = GROUP chars by c;  
grunt> ILLUSTRATE charGroup;
```

chars	c:chararray	
	c	
	c	

charGroup	group:chararray	chars:bag{:tuple(c:chararray)}
	c	{{(c), (c)}}

Inner Bag

Outer Bag

Inner y Outer Bags

```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS  
  (c:chararray);  
grunt> charGroup = GROUP chars by c;  
grunt> dump charGroup;  
(a, { (a), (a), (a) })  
(c, { (c), (c) })  
(i, { (i), (i), (i) })  
(k, { (k), (k), (k), (k) })  
(l, { (l), (l) })
```



Inner Bag

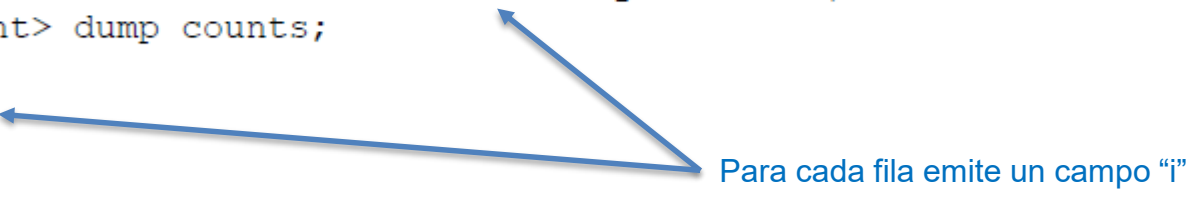


Outer Bag

Pig Latin - Foreach

- FOREACH <bag> GENERATE <data>:
 - Itera sobre cada elemento de un bag y produce un resultado
 - Ejemplo: *grunt> result = foreach bag generate f1;*

```
grunt> records = LOAD 'data/a.txt' AS (c:chararray, i:int);
grunt> dump records;
(a,1)
(d,4)
(c,9)
(k,6)
grunt> counts = foreach records generate i;
grunt> dump counts;
(1)
(4)
(9)
(6)
```




Para cada fila emite un campo "i"

Foreach con funciones

- FOREACH <bag> GENERATE group, FUNCTION(A):
 - Pig incluye algunas de las funciones más comunes: count, flatten, concat,...
 - Se pueden implementar funciones custom

```
grunt> chars = LOAD 'data/b.txt' AS (c:chararray);
grunt> charGroup = GROUP chars by c;
grunt> dump charGroup;
(a,{(a),(a),(a)})
(c,{(c),(c)})
(i,{(i),(i),(i)})
(k,{(k),(k),(k),(k)})
(l,{(l),(l)})
grunt> describe charGroup;
charGroup: {group: chararray, chars: {(c: chararray)}}
grunt> counts = FOREACH charGroup GENERATE group, COUNT(chars);
grunt> dump counts;
(a,3)
(c,2)
(i,3)
(k,4)
(l,2)
```



Para cada fila en el Bag chargroup, emite un campo group y cuenta el número de ítems en el bag "chars"

Funcion Tokenize

- Separa una cadena en tokens y los incluye en una bag
 - Los separadores son: espacio (), comillas dobles (“), coma (,), paréntesis (()) o asterisco (*).

```
grunt> linesOfText = LOAD 'data/c.txt' AS (line:chararray);
grunt> dump linesOfText;
(this is a line of text)
(yet another line of text)
(third line of words)
```

← Separa cada línea por sus espacios y devuelve un bag de tokens.

```
grunt> tokenBag = FOREACH linesOfText GENERATE TOKENIZE(line);
```

```
grunt> dump tokenBag;
(({this),(is),(a),(line),(of),(text)}))
(({yet),(another),(line),(of),(text)}))
(({third),(line),(of),(words)}))
grunt> describe tokenBag;
tokenBag: {bag_of_tokenTuples: {tuple_of_tokens: (token: chararray)}}
```

← Cada fila es un bag de palabras generado por la función TOKENIZE

Operador Flatten

- “Aplana” bags anidados y tipos de datos
- No es una función, es un operador

```
grunt> dump tokenBag;
({ (this), (is), (a), (line), (of), (text) })
({ (yet), (another), (line), (of), (text) })
({ (third), (line), (of), (words) })
grunt> flatBag = FOREACH tokenBag GENERATE flatten($0);
grunt> dump flatBag;
(this)
(is)
(a)
...
...
(text)
(third)
(line)
(of)
(words)
```

Estructura anidada: bag de bags de tuplas

Cada fila se aplana resultando en un bag de tokens simples

Los elementos de un bag se pueden referenciar mediante un índice

Inner Join

```
--InnerJoin.pig
posts = load '/training/data/user-posts.txt' using PigStorage(',')
      as (user:chararray,post:chararray,date:long);
likes = load '/training/data/user-likes.txt' using PigStorage(',')
      as (user:chararray,likes:int,date:long);

userInfo = join posts by user, likes by user;

dump userInfo;
```

Lee los registros del input 1 y los carga en un bag

Lee los registros del input 2 y los carga en un bag

Usa la coma como separador

Une los datasets

Cuando una clave es igual en los dos datasets, las filas se unen en una única fila. En este caso, cuando el nombre es igual.

Inner Join

- Join reutiliza los nombres de los campos de entrada y precede el nombre del bag de entrada
 - `<bag_name>:: <field_name>`

```
grunt> describe posts;
posts: {user: chararray, post: chararray, date: long}
grunt> describe likes;
likes: {user: chararray, likes: int, date: long}
```

```
grunt> describe userInfo; ← Mostrar schema del bag resultante
UserInfo: {
  posts::user: chararray,
  posts::post: chararray,
  posts::date: long,
  likes::user: chararray,
  likes::likes: int,
  likes::date: long}
```

Campos del join que proceden de posts


Campos del join que proceden de likes

Outer Join

```
--LeftOuterJoin.pig
posts = load '/training/data/user-posts.txt'
        using PigStorage(',')
        as (user:chararray,post:chararray,date:long);

likes = load '/training/data/user-likes.txt'
        using PigStorage(',')
        as (user:chararray,likes:int,date:long);

userInfo = join posts by user left outer, likes by user;
dump userInfo;
```



Los registros del bag posts estarán en los resultados aún cuando no haya un match en la bag likes

Outer Join

```
--RightOuterJoin.pig
posts = LOAD '/training/data/user-posts.txt'
        USING PigStorage(',')
        AS (user:chararray,post:chararray,date:long);
likes = LOAD '/training/data/user-likes.txt'
        USING PigStorage(',')
        AS (user:chararray,likes:int,date:long);
userInfo = JOIN posts BY user RIGHT OUTER, likes BY user;
DUMP userInfo;

--FullOuterJoin.pig
posts = LOAD '/training/data/user-posts.txt'
        USING PigStorage(',')
        AS (user:chararray,post:chararray,date:long);
likes = LOAD '/training/data/user-likes.txt'
        USING PigStorage(',')
        AS (user:chararray,likes:int,date:long);
userInfo = JOIN posts BY user FULL OUTER, likes BY user;
DUMP userInfo;
```

Cogroup

- Hace un join de dos datasets preservando la estructura de los dos
- Crea una tupla para cada clave
 - Las tuplas de ambas relaciones que coinciden serán campos

```
--Cogroup.pig
posts = LOAD '/training/data/user-posts.txt'
        USING PigStorage(',')
        AS (user:chararray,post:chararray,date:long);
likes = LOAD '/training/data/user-likes.txt'
        USING PigStorage(',')
        AS (user:chararray,likes:int,date:long);
userInfo = COGROUP posts BY user, likes BY user;
DUMP userInfo;
```


Cogroup

- COGROUP por defecto es un OUTER JOIN
- Se pueden eliminar los registros vacíos con bags vacíos lanzando un INNER en cada Bag
 - Se trata de una INNER JOIN “como funcionalidad”

```
--CogroupInner.pig
posts = LOAD '/training/data/user-posts.txt'
        USING PigStorage(',')
        AS (user:chararray,post:chararray,date:long);
likes = LOAD '/training/data/user-likes.txt'
        USING PigStorage(',')
        AS (user:chararray,likes:int,date:long);
userInfo = COGROUP posts BY user INNER, likes BY user INNER;
DUMP userInfo;
```

UDF

- Pig proporciona la capacidad de definir funciones propias
 - Filter
 - Ex: `res = FILTER bag BY udfFilter(post);`
 - Load Function
 - Ex: `res = load 'file.txt' using udfLoad();`
- Eval
 - Ex: `res = FOREACH bag GENERATE udfEval($1)`
- Choice between several programming languages
 - Java, Python, Javascript
- Pasos:
 - Extender la función
 - Registrar el jar en el script
 - Usar la función en el script

UDF

Crear una función definida por el usuario mediante Java:

```
public class IsShort extends FilterFunc{
    private static final int MAX_CHARS = 15;

    @Override
    public Boolean exec(Tuple tuple) throws IOException {
        if ( tuple == null || tuple.isNull() || tuple.size() == 0 ){
            return false;
        }
        Object obj = tuple.get(0);
        if ( obj instanceof String){
            String st = (String)obj;
            if ( st.length() > MAX_CHARS ){
                return false;
            }
            return true;
        }
        return false;
    }
}
```

Extender FilterFunc e implementar la función exec

Por defecto, un campo único en una tupla

El tipo de Pig CHARARRAY se convertirá en un String

Filtra Strings con una longitud < a 15 caracteres

Los objetos que no sean un String no se incluirán en los resultados

UDF

Registrar la función una vez compilada y empaquetada en un jar:

REGISTER HadoopSamples.jar

- Debe incluir la ruta local al fichero jar.
- La ruta puede ser absoluta o relativa al directorio de ejecución
- La ruta no se debe introducir entre comillas
- Añadirá el fichero jar al Classpath de Java

UDF

Utilizando las funciones definidas:

- Pig localiza funciones buscando en el classpath de de Java el nombre cualificado de la clase.

```
filtered = FILTER posts BY pig.IsShort(post);
```

- Pig distribuirá el Jar registrado y lo registrará en su Classpath
- Es posible crear un alias para una función usando el oferador DEFINE

```
DEFINE isShort pig.IsShort();  
...  
...  
filtered = FILTER posts BY isShort(post);
```

UDF

Utilizando las funciones definidas:

```
--CustomFilter.pig
REGISTER HadoopSamples.jar
DEFINE isShort pig.IsShort();
```

El Jar incluye las funciones de usuario, de este modo son accesibles desde el script

Define un alias "corto" para acceder a la función

```
posts = LOAD '/training/data/user-posts.txt'
        USING PigStorage(',')
        AS (user:chararray,post:chararray,date:long);
```

El script define un esquema: el campo post será del tipo chararray

```
filtered = FILTER posts BY isShort(post);
dump filtered;
```

UDF

Utilizando las funciones definidas:

```
$ hdfs dfs -cat /training/data/user-posts.txt
user1,Funny Story,1343182026191
user2,Cool Deal,1343182133839
user4,Interesting Post,1343182154633
user5,Yet Another Blog,13431839394
```

```
$ pig pig/scripts/CustomFilter.pig
(user1,Funny Story,1343182026191)
(user2,Cool Deal,1343182133839)
```

Los posts cuya longitud supere los 15 caracteres serán filtrados.