
Pig

Fernando Anselmo

<http://fernandoanselmo.orgfree.com/wordpress/>

Versão 1.01 em 12 de novembro de 2023

Resumo

Pig[1] ou "Apache Pig" (aqui chamarei apenas de Pig) é parte do Ecosistema Hadoop criado para ser uma linguagem de script usada para explorar largos datasets. *Pig Latin* é uma extensão do Hadoop, foi desenvolvida pelo Yahoo, para simplificar a programação em alto nível de processamento.

1 Parte inicial

Pig é uma plataforma ou ferramenta de alto nível que é usada para processar grandes conjuntos de dados. Fornece um alto nível de abstração para processamento no *MapReduce* e uma linguagem de script de alto nível, conhecida como *Pig Latin*, que é usada para desenvolver os códigos de análise de dados.



Figura 1: Logo do Apache Pig

No início de 2006, Pig foi desenvolvido pelos pesquisadores do Yahoo. Naquela época, a principal ideia para desenvolver uma ferramenta para executar os trabalhos MapReduce em conjuntos de dados extremamente grandes (mesma ideia e mesmo objetivo do Apache Hive). No ano de 2007, mudou-se para Apache Software Foundation (ASF) como um projeto de código aberto. A primeira versão (0.1) veio no ano de 2008.

Componentes do Pig:

- *Pig Latin* - Linguagem utilizada para expressar fluxos de dados.
- *Pig Engine* - O motor no topo do hadoop

Para processar os dados armazenados no HDFS, escrevemos *scripts* usando a *Pig Latin Language*. Internamente o *Pig Engine* (motor do Pig) converte todos esses *scripts* em um mapa específico e tarefa de redução. Mas estes não ficam visíveis para os programadores por fornecerem um alto nível de abstração. Pig Latin e Pig Engine são os dois principais componentes da ferramenta Apache Pig. O resultado do Pig sempre armazenado no HDFS.

Aplicações do Pig:

- Para explorar grandes conjuntos de dados, o Pig Scripting é usado.
- Fornecer suporte em grandes conjuntos de dados para consultas Ad-hoc.
- Na prototipagem de algoritmos de processamento de grandes conjuntos de dados.
- Processar as cargas de dados sensíveis ao tempo.
- coletar grandes quantidades de conjuntos de dados na forma de logs de pesquisa e rastreamentos da web.
- Em respostas analíticas quando é necessário usar amostragem.

Tipos de modelos de dados no Pig:

- *Atom*: Valor de dados atômicos que é usado para armazenar como uma string. O principal uso deste modelo é poder ser usado como um número e também como uma string.
- *Tuple*: Conjunto ordenado dos campos.
- *Bag*: Coleção de tuplas.
- *Map*: Conjunto de pares chave/valor.

2 Hadoop no Docker

O modo mais simples de se conseguir trabalhar com o Hadoop é utilizando o Docker, para baixar a imagem do Hadoop:

```
$ docker pull suhothayan/hadoop-spark-pig-hive:2.9.2
```

Nessa imagem temos outros produtos do Ecossistema Hadoop: Spark, Pig e Hive. Para criar e executar a primeira vez o contêiner (a pasta que este comando for executado será associada a uma pasta interna chamada **/home/tsthadoop**):

```
$ docker run -it -d --name meu-hadoop -v $(pwd):/home/tsthadoop
suhothayan/hadoop-spark-pig-hive:2.9.2
```

Uma vez interrompido o contêiner:

```
$ docker stop meu-hadoop
```

Podemos executá-lo novamente com os seguintes comandos:

```
$ docker start meu-hadoop
$ docker exec -it meu-hadoop /etc/bootstrap.sh bash
```

2.1 Erro de Permissão

Na primeira vez que entramos é dado um erro na execução do script "bootstrap.sh" de permissão negada para executar o script "spark-env.sh", vamos corrigir isso com o comando:

```
# chmod 777 /usr/local/spark/conf/spark-env.sh
```

Vamos sair do bash:

```
# exit
```

Podemos executá-lo novamente:

```
$ docker exec -it meu-hadoop /etc/bootstrap.sh bash
```

E o erro desapareceu.

E podemos verificar a versão do Hadoop e Pig que estão instalados:

```
$ hadoop version
```

```
$ pig --version
```

O endereço da pasta do Pig:

```
$ cd /usr/local/pig
```

2.2 Modos de Execução

Pig pode ser executado de duas maneiras diferentes:

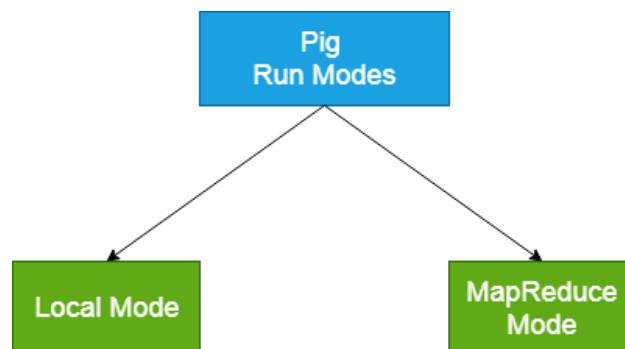


Figura 2: *Modos de Execução*

Modo local - executado em uma única JVM e é usado para experimentação e prototipagem de desenvolvimento. Desta maneira os arquivos são instalados e executados usando localhost. O modo local funciona em um sistema de arquivos local. Os dados de entrada e saída armazenados no sistema de arquivos local.

```
$ pig -x local
```

Modo MapReduce - também é conhecido como modo Hadoop. É o modo padrão. Neste Pig renderiza Pig Latin em trabalhos MapReduce e os executa no cluster. Pode ser executado em uma instalação do Hadoop semi-distribuída ou totalmente distribuída. Aqui, os dados de entrada e saída estão presentes no HDFS.

```
$ pig
```

Ou então

```
$ pig -x mapreduce
```

Para executar um Script em Pig temos as seguintes maneiras:

- **Modo Interativo** - Neste modo, o Pig é executado no shell do Grunt. Para invocar o shell Grunt, execute o comando `pig`. Uma vez que o modo Grunt é executado, podemos fornecer instruções e comandos Pig Latin interativamente na linha de comando.
- **Modo em Lote** - Neste modo, podemos executar um arquivo de script com extensão `.pig`. Esses arquivos contêm comandos do Pig Latin.
- **Modo Embutido** - Neste modo, podemos definir nossas próprias funções. Essas funções podem ser chamadas como UDF (Funções Definidas pelo Usuário). Aqui, usamos linguagens de programação como Java e Python.

2.3 Pig Latin

Sabemos que a *Pig Latin* é uma linguagem de fluxo de dados usada para analisar os dados no Hadoop. É uma linguagem textual que abstrai a programação do idioma Java MapReduce em uma notação.

Tipos de Dados:

- **int** - Inteiro de 32-bit. Exemplo: 2
- **long** - Inteiro de 64-bit. Exemplo: 2L ou 2l
- **float** - Numérico flutuante de 32-bit. Exemplo: 2.5F, 2.5f, 2.5e2f ou 2.5.E2F
- **double** - Numérico flutuante de 64-bit. Exemplo: 2.5, 2.5, 2.5e2 ou 2.5.E2
- **chararray** - Array de caracteres em formato Unicode UTF-8. Exemplo: "Fernando"
- **bytearray** - Array de bytes.
- **boolean** - Tipo lógico que pode conter os valores true/false.
- **datetime** - Tipo data/hora no formato padrão. Exemplo: 2022-03-02T12:54:00.000+00:00
- **biginteger** - Tipo Java BigInteger para inteiros gigantes.
- **bigdecimal** - Tipo Java BigDecimal para numéricos flutuantes gigantes.

Como dito anteriormente, temos ainda os tipos complexos:

- **tuple** - Um conjunto ordenado de campos. Exemplo: (15,12)
- **bag** - Uma coleção de tuplas. Exemplo: {(15,12), (15,22)}
- **map** - Um conjunto de chaves e valores. Exemplo: [chv1#valor]

3 Pig com o HDFS

Antes de fazermos qualquer coisa no Pig precisamos ativar o JobHistory (que por padrão está inativo):

```
# ./usr/local/hadoop-2.9.2/sbin/mr-jobhistory-daemon.sh --config /usr/local/hadoop/etc
start historyserver
```

Para praticarmos um pouco temos um arquivo CSV (que foi disponibilizado na pasta dados em <https://github.com/fernandoans/publicacoes/tree/master/Apostilas/Pig/dados>) chamado SalesJan2009.csv que representa um conjunto de vendas. Nosso primeiro trabalho será colocá-lo a disposição do HDFS.

Colocar o arquivo na pasta associativa do contêiner do Docker (lembramos que existe uma pasta interna que faz as associações quando o contêiner é ativo). Em seguida dentro do contêiner acesse a pasta associativa:

```
# cd /home/tsthadoop
```

Se verificarmos o conteúdo da pasta:

```
# ls
```

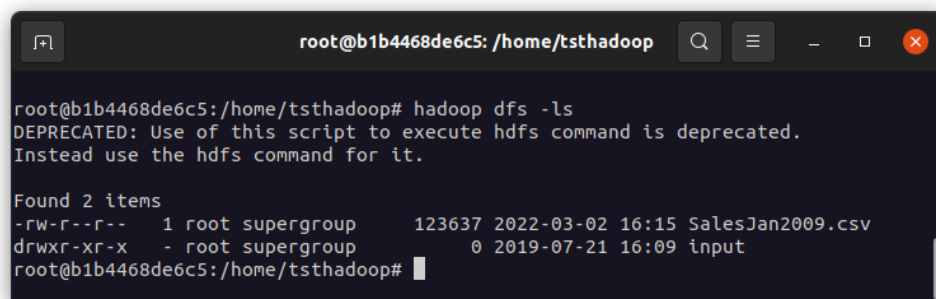
O arquivo CSV indicado deve estar nela. Para adicioná-lo ao HDFS:

```
# hadoop fs -put SalesJan2009.csv
```

E verificamos se o arquivo foi adicionado ao HDFS:

```
# hadoop dfs -ls
```

Que deve mostrar o seguinte resultado:

A terminal window with a dark background. The title bar shows 'root@b1b4468de6c5: /home/tsthadoop'. The command 'hadoop dfs -ls' has been executed. The output shows a deprecation warning and then lists two items: 'SalesJan2009.csv' and 'input'.

```
root@b1b4468de6c5:/home/tsthadoop# hadoop dfs -ls
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 2 items
-rw-r--r--  1 root supergroup      123637 2022-03-02 16:15 SalesJan2009.csv
drwxr-xr-x  - root supergroup         0 2019-07-21 16:09 input
root@b1b4468de6c5:/home/tsthadoop#
```

Figura 3: *Arquivo no HDFS*

Ativamos o Pig em modo MapReduce:

```
# pig
```

No Grunt que é comando prompt do Pig. Executamos o seguinte comando para ler os dados:

```
grunt> salesTable = LOAD 'SalesJan2009.csv' USING PigStorage(',') AS
(Transaction_date:chararray,Product:chararray,Price:chararray,
Payment_Type:chararray,Name:chararray,City:chararray,State:chararray,
Country:chararray,Account_Created:chararray,Last_Login:chararray,
Latitude:chararray,Longitude:chararray);
```

Criar um grupo por País:

```
grunt> GroupByCountry = GROUP salesTable BY Country;
```

Para cada tupla em 'GroupByCountry', gerar uma string resultante do formulário-*Nome do País:*
Nº de produtos vendidos:

```
grunt> CountByCountry = FOREACH GroupByCountry
GENERATE CONCAT((chararray)$0,CONCAT(':', (chararray)COUNT($1)));
```

Armazenar o resultado em um Fluxo de Dados chamado "pig_saida_vendas":

```
grunt> STORE CountByCountry INTO 'pig_saida_vendas' USING PigStorage('\t');
```

E quando esse último comando for colocado a mágica começa, e um MapReduce é iniciado realizando todo o trabalho, pensando no Hadoop isso evita de termos de pensar em criar Scripts para realizar o Map/Reduce.

Saímos do Pig com CTRL+C e damos o seguinte comando para vermos o resultado:

```
# hdfs dfs -cat pig_saida_vendas/part-r-00000
```

Outra forma de visualizarmos os resultados é através do navegador acessando o HDFS em <http://localhost:50070>, na página acessar no menu superior as opções: Utilities — Browse the file system. Na página navegar para o diretório: /user/root e temos o seguinte resultado:

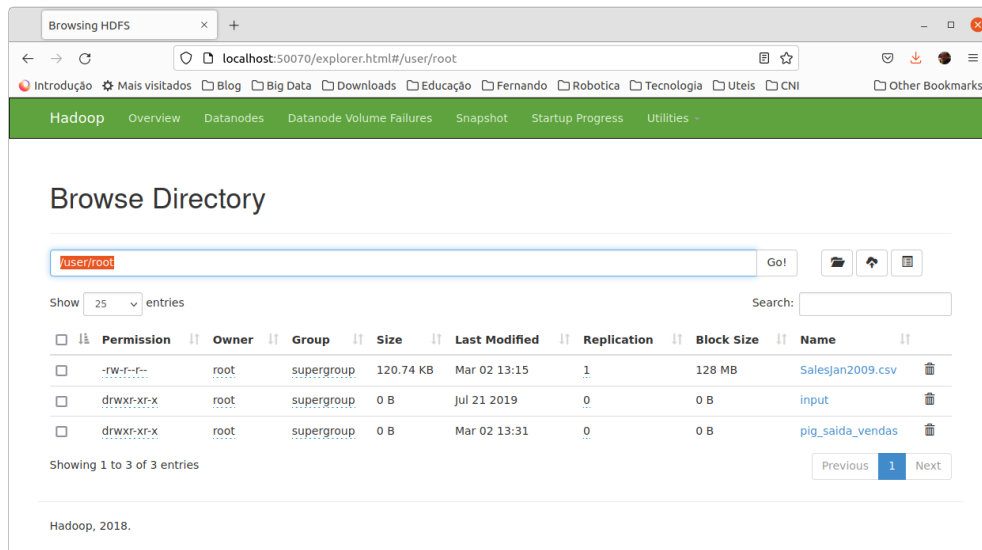


Figura 4: Arquivo no HDFS no Navegador

Clicar em pig_saida_vendas e temos o arquivo que foi gerado. Percebeu que podemos fazer muita coisa nessa página como adicionar ou remover arquivos e pastas ou navegar por todas as pastas existentes no HDFS. Aconselho que explore bem essa página e suas funcionalidades.

3.1 Pig Script

Outra forma de fazermos o mesmo trabalho seria a criação de um Script, para isso, digite o seguinte comando:

```
# vim pigvenda.pig
```

Digite "i" (para entrar em modo de inserção) e insira a seguinte codificação:

```
1 salesTable = LOAD 'SalesJan2009.csv' USING PigStorage(',') AS
    (Transaction_date:chararray, Product:chararray, Price:chararray,
    Payment_Type:chararray, Name:chararray, City:chararray, State:chararray,
    Country:chararray, Account_Created:chararray, Last_Login:chararray,
    Latitude:chararray, Longitude:chararray);
2 GroupByCountry = GROUP salesTable BY Country;
3 CountByCountry = FOREACH GroupByCountry GENERATE
    CONCAT((chararray)$0,CONCAT(':', (chararray)COUNT($1)));
4 STORE CountByCountry INTO 'pig_saida2_vendas' USING PigStorage('\t');
```

Para salvar pressionar ESC (para sair do modo de inserção) e ":wq" para salvar e sair. Para executar:

```
# pig pigvenda.pig
```

Fantástico isso não, as mesmas ações serão executadas só que agora através de um Script que pode ser guardado e executado para futuras modificações no arquivo. Verifique que novamente foi gerado o arquivo de saída (agora com outro nome):

```
# hdfs dfs -cat pig_saida2_vendas/part-r-00000
```

Como diria Jack vamos por partes. Então em cada subseção abaixo vamos nos concentrar somente no operador em questão para entender o que cada um deles realiza, e assim termos uma ideia do que podemos obter com Pig.

3.2 Operador Load

Utilizado para ler dados do sistema de arquivos HDFS, sua sintaxe é:

```
LOAD 'arquivo' [USING tipo] [AS estrutura];
```

Por exemplo, vamos supor que temos o seguinte arquivo (criar com o nome notas.txt e em seguida adicioná-lo ao HDFS):

```
1 Joana,5.0,6.0,7
2 Carlos,7.0,3.0,8
3 Marla,6.5,4.2,7
```

Contendo o nome, notas e total de faltas dos alunos em uma determinada matéria. Desejamos ler esses dados.

```
grunt> tabnotas = LOAD 'notas.txt' USING PigStorage(',') AS (nome:chararray,
nota1:double, nota2:double, faltas:int);
```

Vamos executar esse comando:

```
grunt> DUMP tabnotas;
```

E verificar o resultado:

```
grunt> DESCRIBE tabnotas;
```

Obviamente é bem mais interessante utilizar esse operador em combinação com outros.

3.3 Operador Store

Utilizado para armazenar ou salvar resultados no sistema de arquivos HDFS, sua sintaxe é:

```
STORE arquivo INTO 'diretorio' [USING funcao];
```

Por exemplo, vamos supor o arquivo "notas.txt" criado anteriormente. Lemos seus dados.

```
grunt> tabnotas = LOAD 'notas.txt' USING PigStorage(',') AS (nome:chararray,
nota1:double, nota2:double, faltas:int);
```

Vamos executar esse comando:

```
grunt> DUMP tabnotas;
```

E podemos armazenar o resultado no HDFS:

```
grunt> STORE tabnotas INTO 'saidaNotas' USING PigStorage('*');
grunt> CAT saidaNotas;
```

E teremos com resultado do processo:

```
1 nome***
2
3 Joana*5.0*6.0*7
4
5 Carlos*7.0*3.0*8
6
7 Marla*6.5*4.2*7
```

Ficou separado por '*' pois foi esse caractere que definimos na função "PigStore(char)".

3.4 Operador Cross

Operador para facilitar a computação para um produto de duas ou mais relações. Isso facilita enormemente o processamento. Para exemplificarmos seu uso precisamos de dois arquivos, "matriz1.txt" com o seguinte conteúdo:

```
1 1,2
2 3,4
```

E "matriz2.txt" com o seguinte conteúdo:

```
1 5,6,7
2 8,9,10
```

Criar ambos arquivos e adicioná-los ao HDFS. Agora no Pig precisamos ler ambos:

```
grunt> arq1 = LOAD 'matriz1.txt' USING PigStorage(',') AS (a1:int,a2:int);
grunt> DUMP arq1;
grunt> arq2 = LOAD 'matriz2.txt' USING PigStorage(',') AS (b1:int,b2:int,b3:int);
grunt> DUMP arq2;
```

E agora podemos realizar o relacionamento cruzado entre eles:

```
grunt> resultado = CROSS arq1,arq2;
grunt> DUMP resultado;
```

E teremos com resultado do processo:

```
1 (1,2,5,6,7)
2 (1,2,8,9,10)
3 (3,4,5,6,7)
4 (3,4,8,9,10)
```


3.5 Operador Distinct

Usado para remover tuplas duplicadas nas relações. Para realizar esse procedimento Pig ordena os dados e elimina os duplicados. Vamos criar o arquivo "dados.txt" com o seguinte conteúdo:

```
1 1,2,3
2 4,5,6
3 1,2,3
4 7,8,9
5 4,5,6
```

Adicioná-lo ao HDFS. Agora no Pig vamos ler o arquivo:

```
grunt> arq = LOAD 'dados.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int);
grunt> DUMP arq;
```

E aplicamos o comando *distinct*:

```
grunt> resultado = DISTINCT arq;
grunt> DUMP resultado;
```

E teremos com resultado do processo:

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

3.6 Operador Filter

Utilizado para remover tuplas da relação que não condizem a um determinado critério, se está tentando associar ao SQL, este operador seria o WHERE. Vamos criar o arquivo "dados.txt" com o seguinte conteúdo:

```
1 1,8
2 2,8
3 2,3
4 8,9
5 5,8
```

Adicioná-lo ao HDFS. Agora no Pig vamos ler o arquivo:

```
grunt> arq = LOAD 'dados.txt' USING PigStorage(',') AS (a1:int,a2:int);
grunt> DUMP arq;
```

E aplicamos o comando *distinct*:

```
grunt> resultado = FILTER arq BY a2==8;
grunt> DUMP resultado;
```

E teremos com resultado do processo:

```
1 (1,8)
2 (2,8)
3 (5,8)
```

3.7 Operador ForEach

Realiza uma interação com cada um dos elementos arrumando-os no formato que desejamos. Vamos criar o arquivo "dados.txt" com o seguinte conteúdo:

```
1 1,2,3
2 4,5,6
3 7,8,9
```

Adicioná-lo ao HDFS. Agora no Pig vamos ler o arquivo:

```
grunt> arq = LOAD 'dados.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int);
grunt> DUMP arq;
```

E aplicamos o comando *foreach*:

```
grunt> resultado = FOREACH arq GENERATE CONCAT('a:', (chararray)a1),
CONCAT('b:', (chararray)a2), CONCAT('c:', (chararray)a3);
grunt> DUMP resultado;
```

E teremos com resultado do processo:

```
1 (a:1,b:2,c:3)
2 (a:4,b:5,c:6)
3 (a:7,b:8,c:9)
```

3.8 Operador Group

Agrupar um ou mais relacionamento entre os campos, se estiver comparando com o SQL acabou de encontrar o operador GROUP BY, porém aqui temos algo mais especializado. Vamos criar o arquivo "familia.txt" com o seguinte conteúdo:

```
1 Fernando,Anselmo
2 Jessica,Rabbit
3 Robin,Batman
4 Hugo,Anselmo
5 Roger,Rabbit
6 Fox,Rabbit
7 Bruce,Batman
```

Adicioná-lo ao HDFS. Agora no Pig vamos ler o arquivo:

```
grunt> arq = LOAD 'familia.txt' USING PigStorage(',') AS (n1:chararray, n2:chararray);
grunt> DUMP arq;
```

E aplicamos o comando *group*:

```
grunt> resultado = group arq by n2;
grunt> DUMP resultado;
```

E teremos com resultado do processo:

```
1 ( Batman,{(Bruce,Batman),(Robin,Batman)})
2 ( Rabbit,{(Fox,Rabbit),(Roger,Rabbit),(Jessica,Rabbit)})
3 ( Anselmo,{(Hugo,Anselmo),(Fernando,Anselmo)})
```

3.9 Operador Limit

Limitar a quantidade das tuplas mostradas, se estiver comparando com o SQL este é o LIMIT (de alguns bancos, pois em outros é TOP), vamos nos aproveitar do arquivo carregado no comando anterior, que ao dar um comando:

```
grunt> DUMP arq;
```

Temos como saída:

```
1 (Fernando,Anselmo)
2 (Jessica,Rabbit)
3 (Robin,Batman)
4 (Hugo,Anselmo)
5 (Roger,Rabbit)
6 (Fox,Rabbit)
7 (Bruce,Batman)
```

E aplicamos o comando *limit*:

```
grunt> resultado = LIMIT arq 3;
grunt> DUMP resultado;
```

Temos como saída:

```
1 (Fernando,Anselmo)
2 (Jessica,Rabbit)
3 (Robin,Batman)
```

3.10 Operador Order

Ordenar as saídas das tuplas mostradas, se estiver comparando com o SQL este é o ORDER BY, vamos nos aproveitar do arquivo carregado no comando GROUP e ordenar sua saída com o comando *order*:

```
grunt> resultado = ORDER arq BY n1 ASC;
grunt> DUMP resultado;
```

Os campos podem ser ordenados de modo descendente (DESC) ou ascendente (ASC). Temos como saída:

```
1 (Fernando,Anselmo)
2 (Jessica,Rabbit)
3 (Robin,Batman)
```

3.11 Operador Split

Cria uma nova saída através de uma condição determinada, porém TODAS as condições de campo devem ser abrangidas, por exemplo: um arquivo que contém 2 gêneros M e F, deve ter uma condição para cada um dos gêneros. Nosso arquivo possui 3 famílias, assim:

```
grunt> SPLIT arq INTO famBatman IF EqualsIgnoreCase(n2,'Batman'),
famRabbit IF EqualsIgnoreCase(n2,'Rabbit'),
```

```
famAnselmo IF EqualsIgnoreCase(n2,'Anselmo');
```

E podemos aplicar o :

```
grunt> DUMP famBatman;
```

E temos como saída a família "Batman":

```
1 (Robin,Batman)
2 (Bruce,Batman)
```

3.12 Operador Union

Se entendeu o comando Split o Union será extremamente fácil de demonstrar, pois faz exatamente o contrário, vamos juntar as famílias Batman e Rabbit em um único resultado:

```
grunt> resultado = UNION famBatman,famRabbit;
```

E temos como saída as famílias unidas:

```
1 (Jessica,Rabbit)
2 (Roger,Rabbit)
3 (Fox,Rabbit)
4 (Robin,Batman)
5 (Bruce,Batman)
```

4 Resumão Geral

Mas afinal de contas qual a vantagem do Pig, pois tudo o que fazemos com ele poderíamos realizar com um SQL no Hive (por exemplo), isso é uma verdade. Porém devemos olhar para o Pig como uma alternativa *Script* que podemos agendar, executar a qualquer momento de um modo simples e prático, basta apenas nos acostumarmos com a sintaxe deste.

Por exemplo, o seguinte SQL que localiza as vendas superiores a 2.000,00 na cidade do Texas, comparativamente em um Script Pig é escrito do seguinte modo:

SQL	Pig
<pre> SELECT c_id , SUM(amount) AS CTotal FROM customers c JOIN sales s ON c.c_id = s.c_id WHERE c.city = 'Texas' GROUP BY c_id HAVING SUM(amount) > 2000 ORDER BY CTotal DESC </pre>	<pre> customer = LOAD '/data/customer.dat' AS (c_id,name,city); sales = LOAD '/data/sales.dat' AS (s_id,c_id,date,amount); salesBLR = FILTER customer BY city == 'Texas'; joined= JOIN customer BY c_id, salesTX BY c_id; grouped = GROUP joined BY c_id; summed= FOREACH grouped GENERATE GROUP, SUM(joined.salesTX::amount); spenders= FILTER summed BY \$1 > 2000; sorted = ORDER spenders BY \$1 DESC; DUMP sorted; </pre>

Figura 5: Comparando SQL e Pig Latin

Mais fontes de informação podem ser obtidas em diversos sites que apresenta tutoriais completos sobre o Apache Pig como a Tutorials Point[2].

5 Conclusão

Pig é uma plataforma de fluxo de dados de alto nível para executar programas MapReduce do Hadoop. A linguagem usada é a *Pig Latin*. Os scripts Pig são convertidos internamente em tarefas Map Reduce e executados em dados armazenados no HDFS. Além disso, Pig também pode executar seu trabalho no Apache Tez ou no Apache Spark.

Pig pode lidar com qualquer tipo de dados, ou seja, estruturados, semiestruturados ou não estruturados e armazena os resultados correspondentes no Hadoop Data File System. Todas as tarefas que podem ser realizadas usando o PIG também podem ser realizadas usando o java usado no MapReduce.

Pig consome menos linha de código para realizar qualquer operação. Seu código é flexível o suficiente para ser reutilizado novamente. Fornece um conceito útil de tipos de dados aninhados, como *tuple*, *bag* e *map*.

Sou um entusiasta do mundo **Open Source** e novas tecnologias. Qual a diferença entre Livre e Open Source? Livre significa que esta apostila é gratuita e pode ser compartilhada a vontade. Open Source além de livre todos os arquivos que permitem a geração desta (chamados de arquivos fontes) devem ser disponibilizados para que qualquer pessoa possa modificar ao seu prazer, gerar novas, complementar ou fazer o que quiser. Os fontes da apostila (que foi produzida com o LaTeX) está disponibilizado no GitHub [5]. Veja ainda outros artigos que publico sobre tecnologia através do meu Blog Oficial [3].

Referências

- [1] Página do Apache Pig
<https://pig.apache.org/>

- [2] Tutorials Point sobre Pig
https://www.tutorialspoint.com/apache_pig/index.htm
- [3] Fernando Anselmo - Blog Oficial de Tecnologia
<http://www.fernandoanselmo.blogspot.com.br/>
- [4] Encontre essa e outras publicações em
<https://cetrex.academia.edu/FernandoAnselmo>
- [5] Repositório para os fontes da apostila
<https://github.com/fernandoans/publicacoes>