Apache Pig

What is Pig ?

- A High Level data flow language
- Best for analyzing Large datasets
- Geared for parallel processing
- Compiler produces a sequence of MR programs (in –x mapreduce)
- Operates on files on Local machine/HDFS
- Metadata is not required, but can be used when available (HCatlog)
- Optimal tool for semi-structured data/ETL needs on Hadoop
- Can also be run on Tez & Spark

Why Pig ?

- Pig follows multi-query approach, which helps combine multiple operation together and hence reducing the number of scans on data
- Its simple and concise: 1/20th the lines of code and 1/16th the dev time compared to MapReduce
- This Pig eats anything: Structured, Semi-Structured and UnStructured. It adds the complex data-types like Bags, Tuples etc., missing in MR
- Its inherently Lazy! – hence can produce efficient execution plan
- Sampling in any use case. Ex: Data Profiling
- Functionalities can be easily extended through UDFs – including Python, Java, Ruby etc.,

Pig v/s Hive

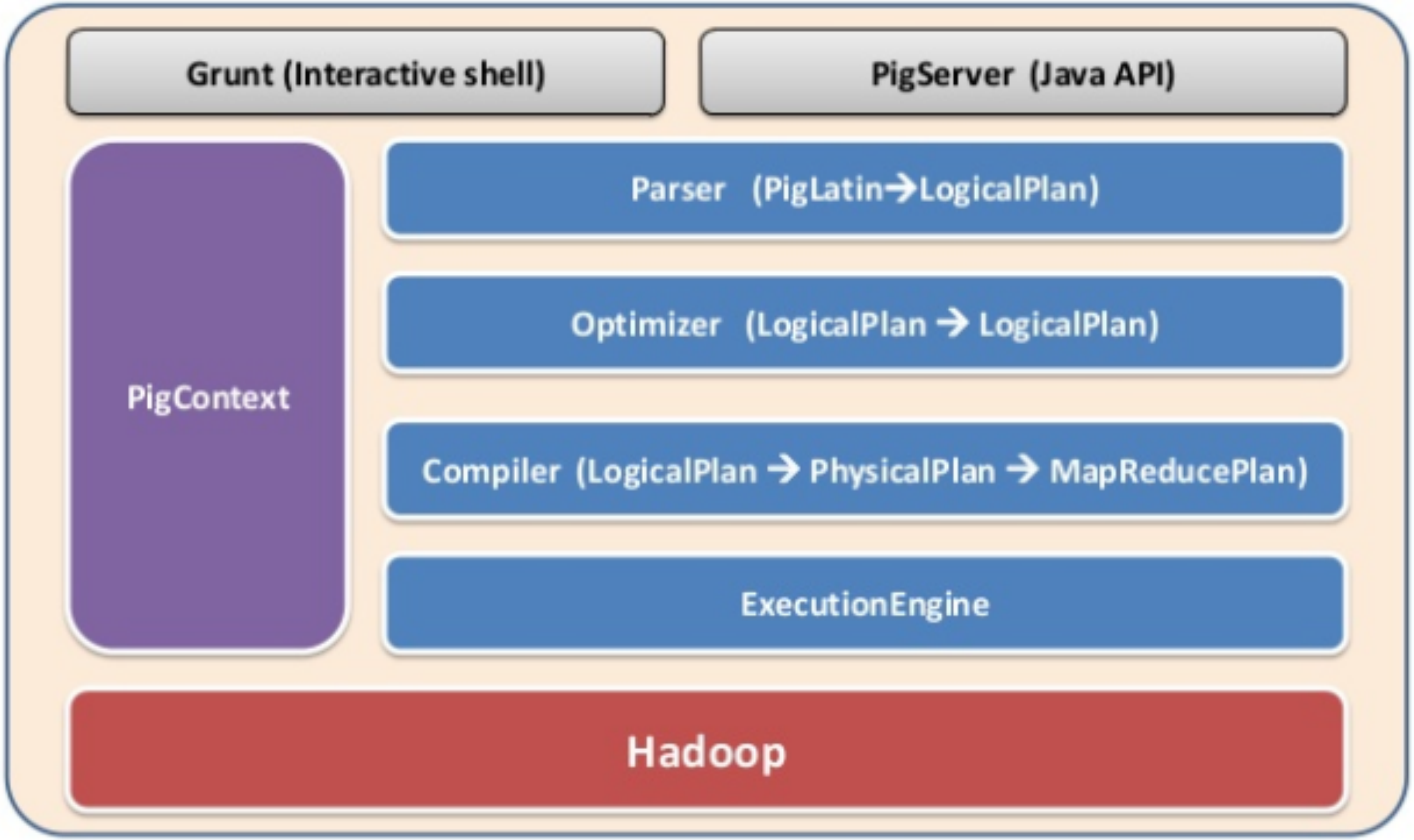
- Hive is SQL like; Pig too – but not really! It's a procedural language
- Pig is used for data analysis and not mainly for creating reports and dashboards like Hive
- They have similar execution times +/- 25% variance
- Pig doesn't need/doesn't have a dedicated metastore unlike Hive. However, it could use one when available.
- Pig operates on client side - Hive on server side
- Pig doesn't support partitions – Hive does
- Pig doesn't support ODBC/JDBC – Hive does

* If you're good at SQL; chances are you'll not use Pig unless needed to ETL unstructured data

‘COMPLEX’ Datatypes

- BAG is collection of TUPLES ~ Tables in our world. Interchangeably called as a RELATION. Ex: ((Amar, 30, M) ,(Vinay, 25, M))
- A TUPLE is an ordered set. Ex: (name, age, sex)
- A FIELD is a piece of data. Ex: `name`
- And then there is MAP. Ex: [‘Name’ # ‘Joydeep’]
- In PIG, Null is UNKNOWN or NON-EXISTENT, similar to SQL though
- There is no string or char; its CHARARRAY

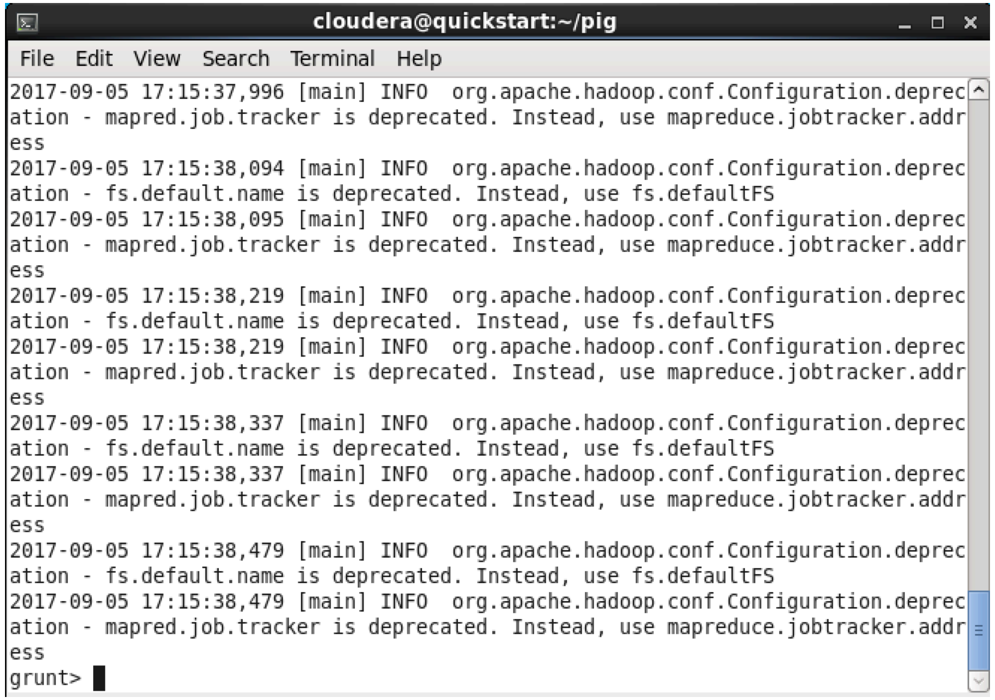
Architecture of Pig


Grunt (Interactive shell)
PigServer (Java API)
PigContext
Parser (PigLatin→LogicalPlan)
Optimizer (LogicalPlan → LogicalPlan)
Compiler (LogicalPlan → PhysicalPlan → MapReducePlan)
ExecutionEngine
Hadoop

Starting PIG

- MapReduce: >pig or >pig –x mapreduce
- Local: >pig –x local
- Tez: >pig –x tez
- Spark: >pig –x spark. 1)doesn’t make sense 2)still not completely available

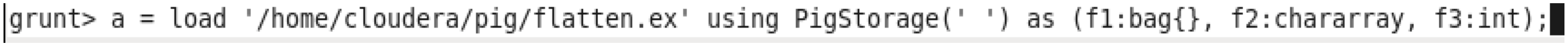
- You should see grunt> shell



```
2017-09-05 17:15:37,996 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2017-09-05 17:15:38,094 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2017-09-05 17:15:38,095 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2017-09-05 17:15:38,219 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2017-09-05 17:15:38,219 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2017-09-05 17:15:38,337 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2017-09-05 17:15:38,337 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
2017-09-05 17:15:38,479 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2017-09-05 17:15:38,479 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
grunt> 
```

Loading Data

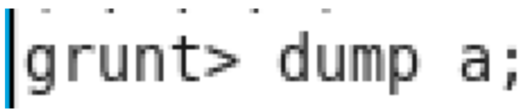
- LOAD



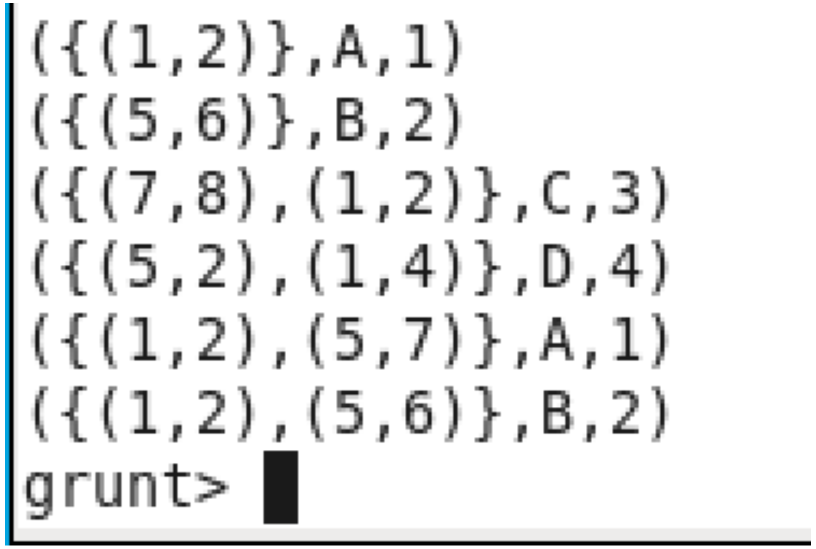
```
grunt> a = load '/home/cloudera/pig/flatten.ex' using PigStorage(' ') as (f1:bag{}, f2:chararray, f3:int);
```

Viewing data (like select)

- DUMP



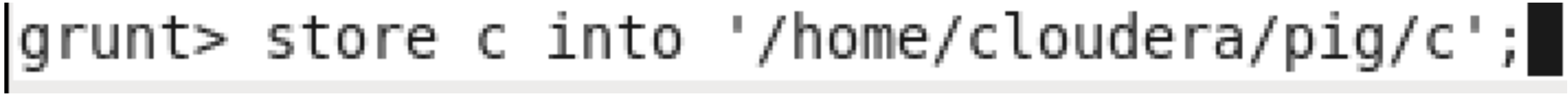
```
grunt> dump a;
```



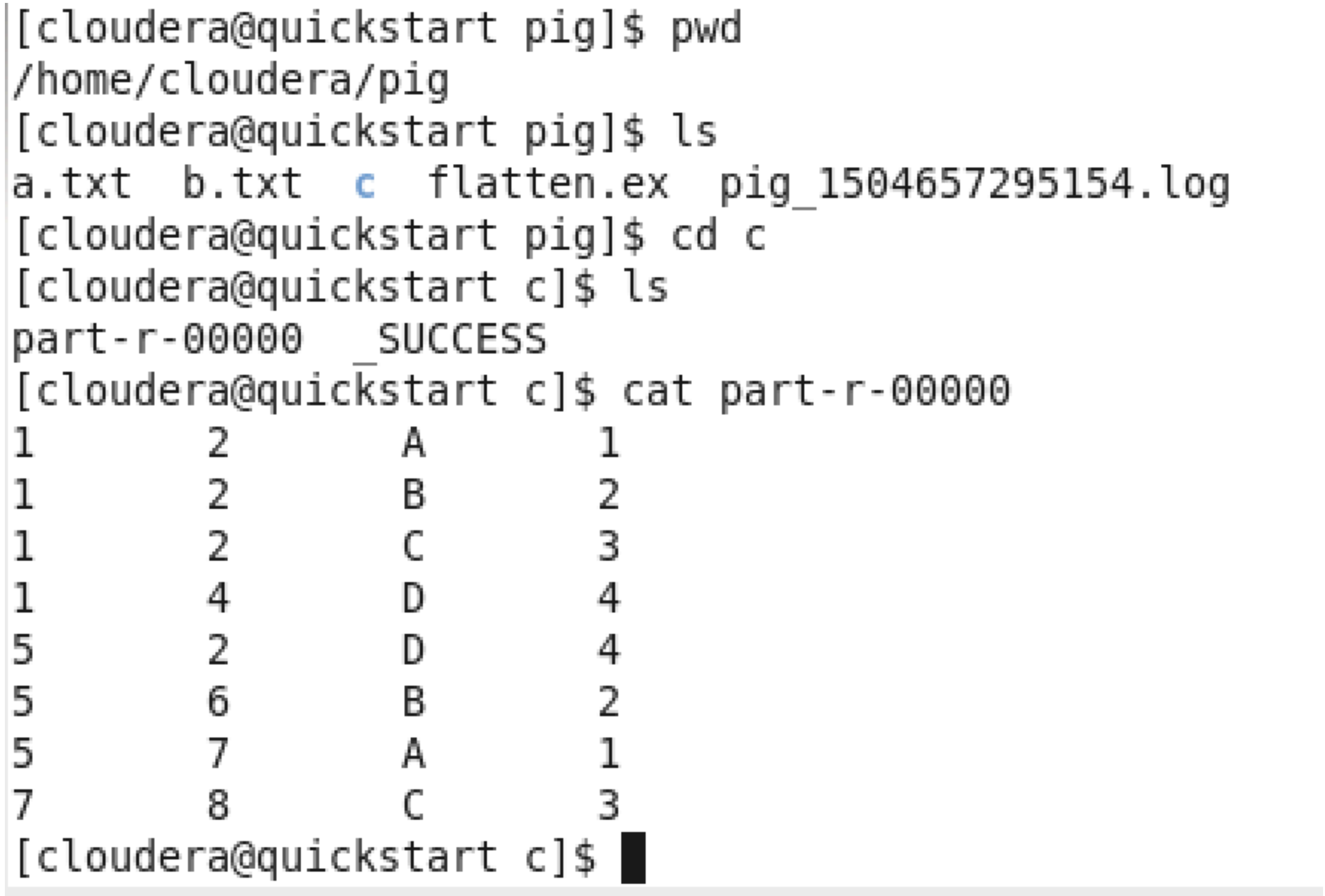
```
({(1,2)},A,1)
({(5,6)},B,2)
({(7,8),(1,2)},C,3)
({(5,2),(1,4)},D,4)
({(1,2),(5,7)},A,1)
({(1,2),(5,6)},B,2)
grunt> 
```

Storing Data

- STORE



```
grunt> store c into '/home/cloudera/pig/c';
```



```
[cloudera@quickstart pig]$ pwd
/home/cloudera/pig
[cloudera@quickstart pig]$ ls
a.txt  b.txt  c  flatten.ex  pig_1504657295154.log
[cloudera@quickstart pig]$ cd c
[cloudera@quickstart c]$ ls
part-r-00000  _SUCCESS
[cloudera@quickstart c]$ cat part-r-00000
1       2       A       1
1       2       B       2
1       2       C       3
1       4       D       4
5       2       D       4
5       6       B       2
5       7       A       1
7       8       C       3
[cloudera@quickstart c]$ 
```

EVERYTHING IN BETWEEN

- Pig is a ‘Data Flow’ language. Implies, each line of code is a step in the pipeline.
    - download script and data files from https://github.com/ravikodi1/pig
    - exec /home/cloudera/pig/script.pig once the extracted folder is placed under /home/cloudera

Important Functions

- FLATTEN
- TOKENIZE
- COUNT vs COUNT_STAR
- PigStorage() vs PigDump() vs BinStorage()
- TOBAG

- supports compression for .gz and .bz

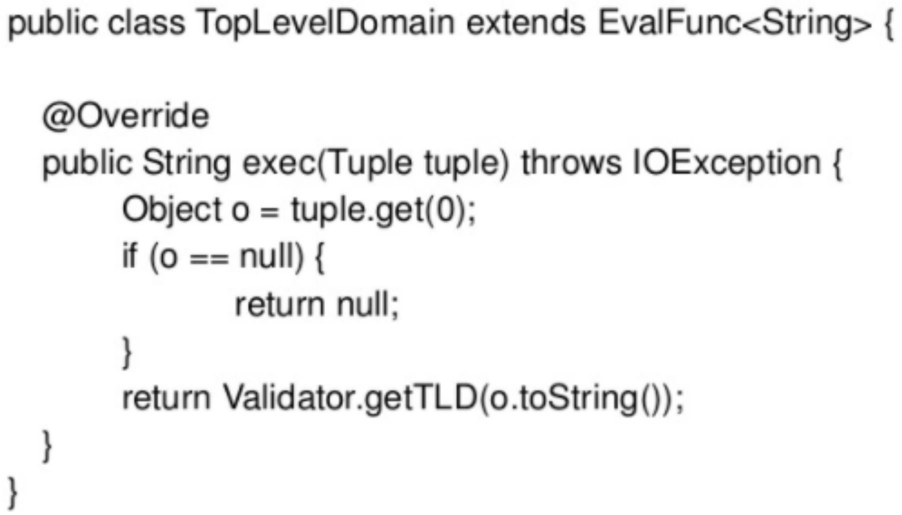
Important Relational Operators

- FOREACH
- SAMPLE
- COGROUP
- SPLIT (Pig doesn't support conditional loops; this is as good as it gets)

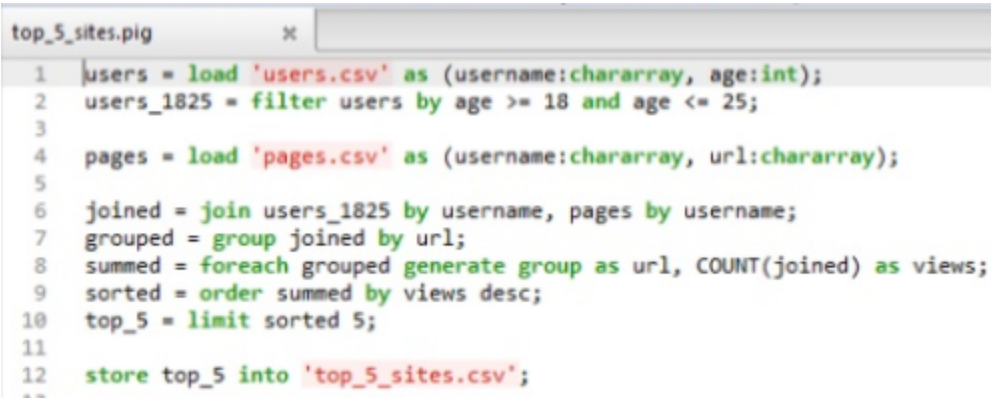
UDFs

custom UDFs can be built using Python, Java, C++, Ruby On Rails.

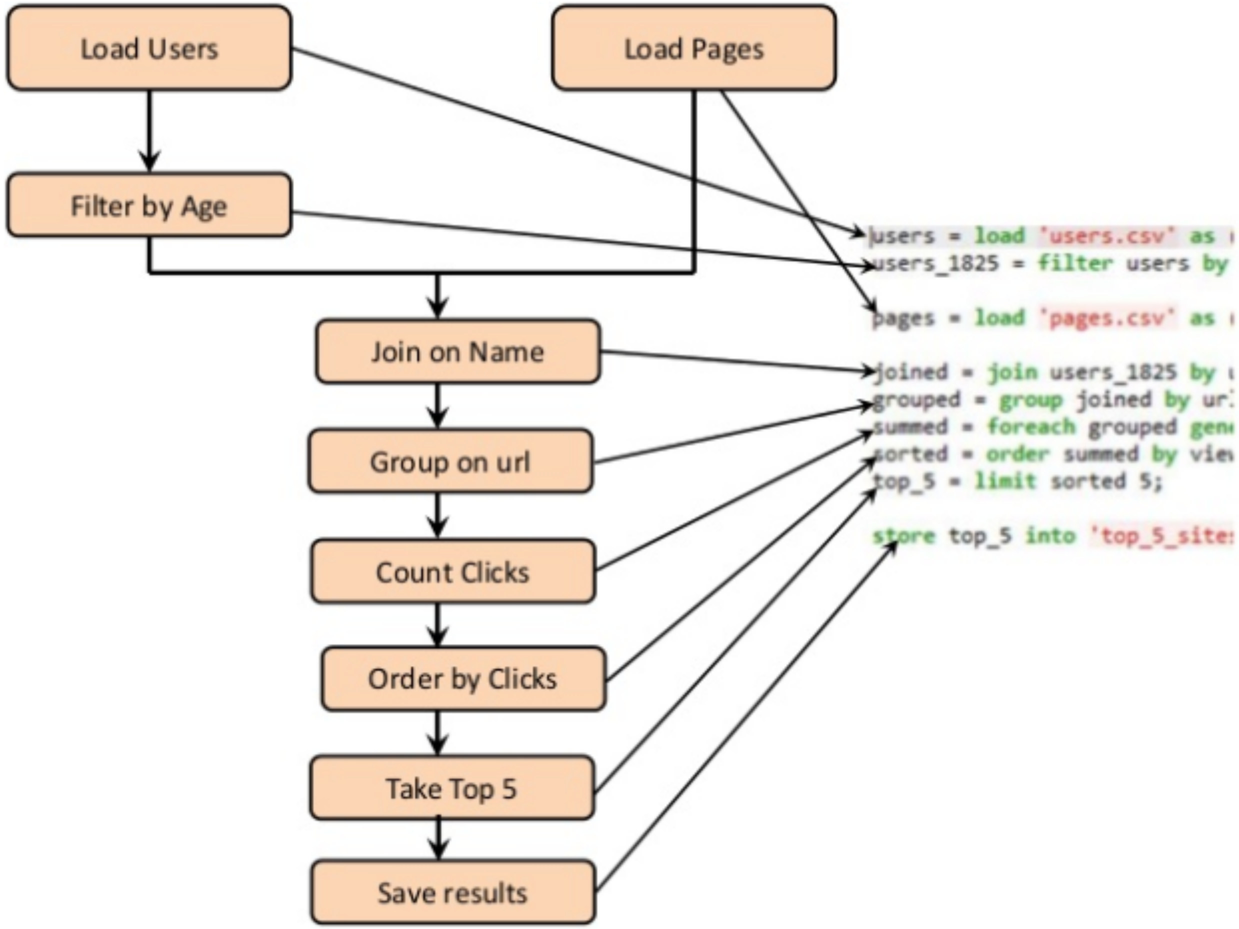
They help extend the functionality of Pig while promoting modularity and reusability of repetitive logic



```
public class TopLevelDomain extends EvalFunc<String> {

   @Override
   public String exec(Tuple tuple) throws IOException {
         Object o = tuple.get(0);
         if (o == null) {
                  return null;
         }
         return Validator.getTLD(o.toString());
   }
}
```



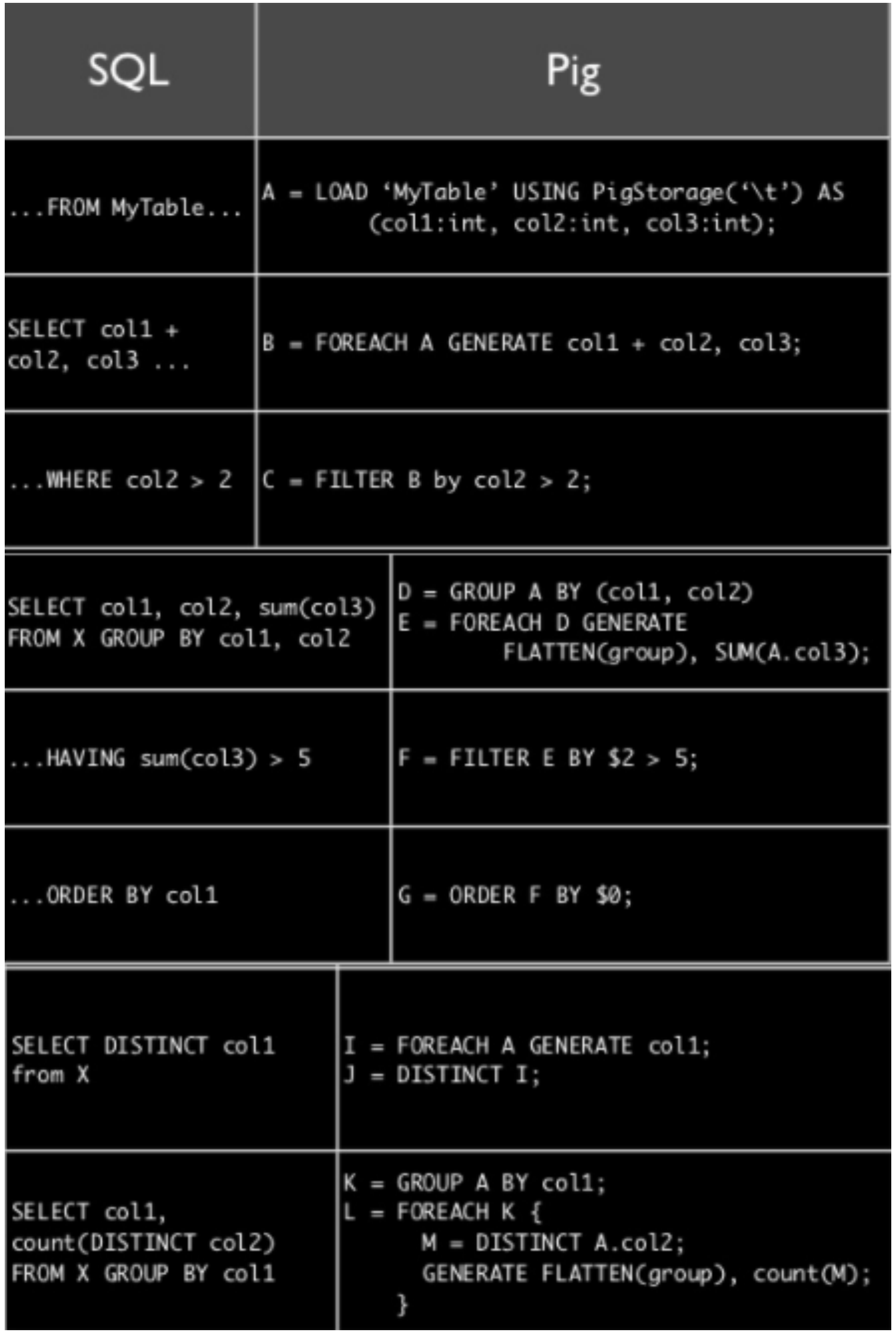
```
users = load 'users.csv' as (username:chararray, age:int);
users_1825 = filter users by age >= 18 and age <= 25;

pages = load 'pages.csv' as (username:chararray, url:chararray);

joined = join users_1825 by username, pages by username;
grouped = group joined by url;
summed = foreach grouped generate group as url, COUNT(joined) as views;
sorted = order summed by views desc;
top_5 = limit sorted 5;

store top_5 into 'top_5_sites.csv';
```



```
users = load 'users.csv' as
users_1825 = filter users by

pages = load 'pages.csv' as

joined = join users_1825 by
grouped = group joined by ur
summed = foreach grouped gen
sorted = order summed by vie
top_5 = limit sorted 5;

store top_5 into 'top_5_site
```

Common SQL statements

and their equivalent PIG representation



SQL	Pig

...FROM MyTable...	A = LOAD ‘MyTable’ USING PigStorage(‘\t’) AS
	(col1:int, col2:int, col3:int);

SELECT col1 + col2, col3 ...	B = FOREACH A GENERATE col1 + col2, col3;
...WHERE col2 > 2	C = FILTER B by col2 > 2;

SELECT col1, col2, sum(col3)
FROM X GROUP BY col1, col2
	D = GROUP A BY (col1, col2)
	E = FOREACH D GENERATE
	FLATTEN(group), SUM(A.col3);

...HAVING sum(col3) > 5	F = FILTER E BY $2 > 5;
...ORDER BY col1	G = ORDER F BY $0;

SELECT DISTINCT col1
from X
	I = FOREACH A GENERATE col1;
	J = DISTINCT I;

SELECT col1,
count(DISTINCT col2)
FROM X GROUP BY col1
	K = GROUP A BY col1;
	L = FOREACH K {
	M = DISTINCT A.col2;
	GENERATE FLATTEN(group), count(M);
	}