LEARN.
CONNECT.
ELEVATE.

YCBS 257 - Data at Scale (Winter 2019)

Instructor: Khaled Tannir

McGill | School of Continuing Studies

mcgill.ca
/continuingstudies

School of Continuing Studies

YCBS 257 - Data at Scale (BIG DATA)

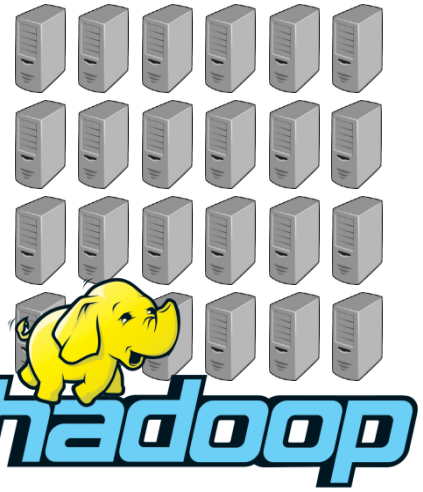
Course 3

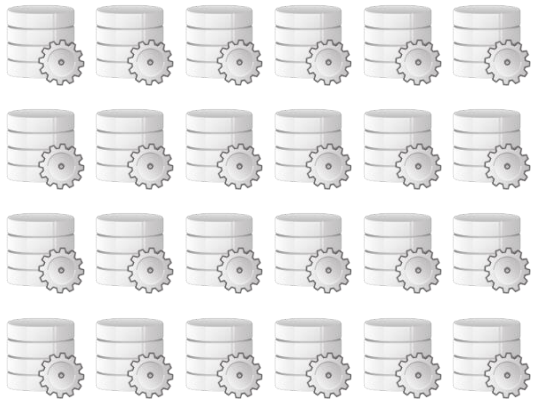
Hadoop Analyzing Tools

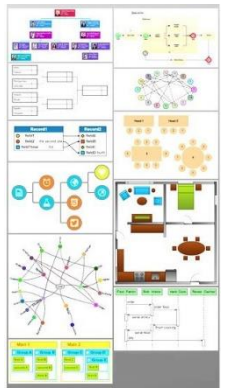
Khaled Tannir

McGill

Machine Learning at Scale







Analyze

Ingest

Theme of this Course

- *Apache PIG*
  - *Core Concepts*
  - *PigLatin script language*

Apache Pig

Making Data Transformation Easy

McGill

FALL 2018

Apache PIG

- Pig is an open source high-level dataflow system
- Originally introduced by Yahoo
- Provides abstraction over MapReduce
- A tool used with Hadoop to analyze (very) large datasets representing them as data flows.
- To write data analysis programs, Pig provides a high-level language known as Pig Latin.

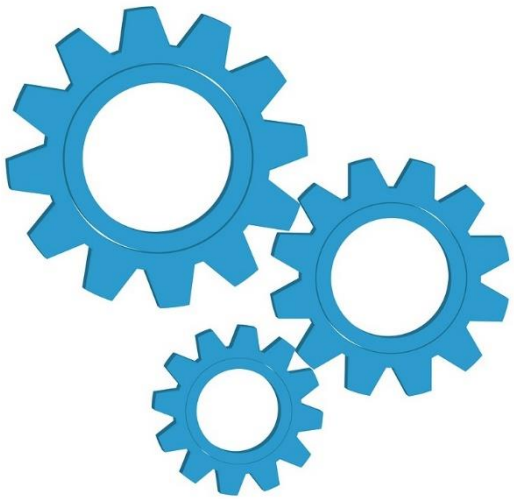
Why do I need Pig ?

- In MapReduce you need to write a program in Java or Python to process the data
- What if you are from non-programing background or you are not a java developer?

- Writing MapReduce program is a real challenge

What Pig Does

- Pig was designed for performing a long series of data operations, making it ideal for three categories of Big Data jobs:
  - Extract-transform-load (ETL) data pipelines,
  - Research on raw data
  - Iterative data processing



Features of PIG

- **Pig has a rich set of common operators to perform operations**
  like join, group, sort, filer, etc.
- **Ease of programing, and provides support for data types**
  long, float, chararray,…
- **Pig is extensible and supports User Defined Functions (*UDF*)**
  UDFs can be created in other programming languages such as Java and invoked in Pig Scripts.
- **Schema not mandatory, but used when available**
- **Handles all kind of data**
  Pig analyzes all kinds of data, structured and unstructured and stores the results in HDFS

Pig Core Components

= Pig Latin + Pig Engine

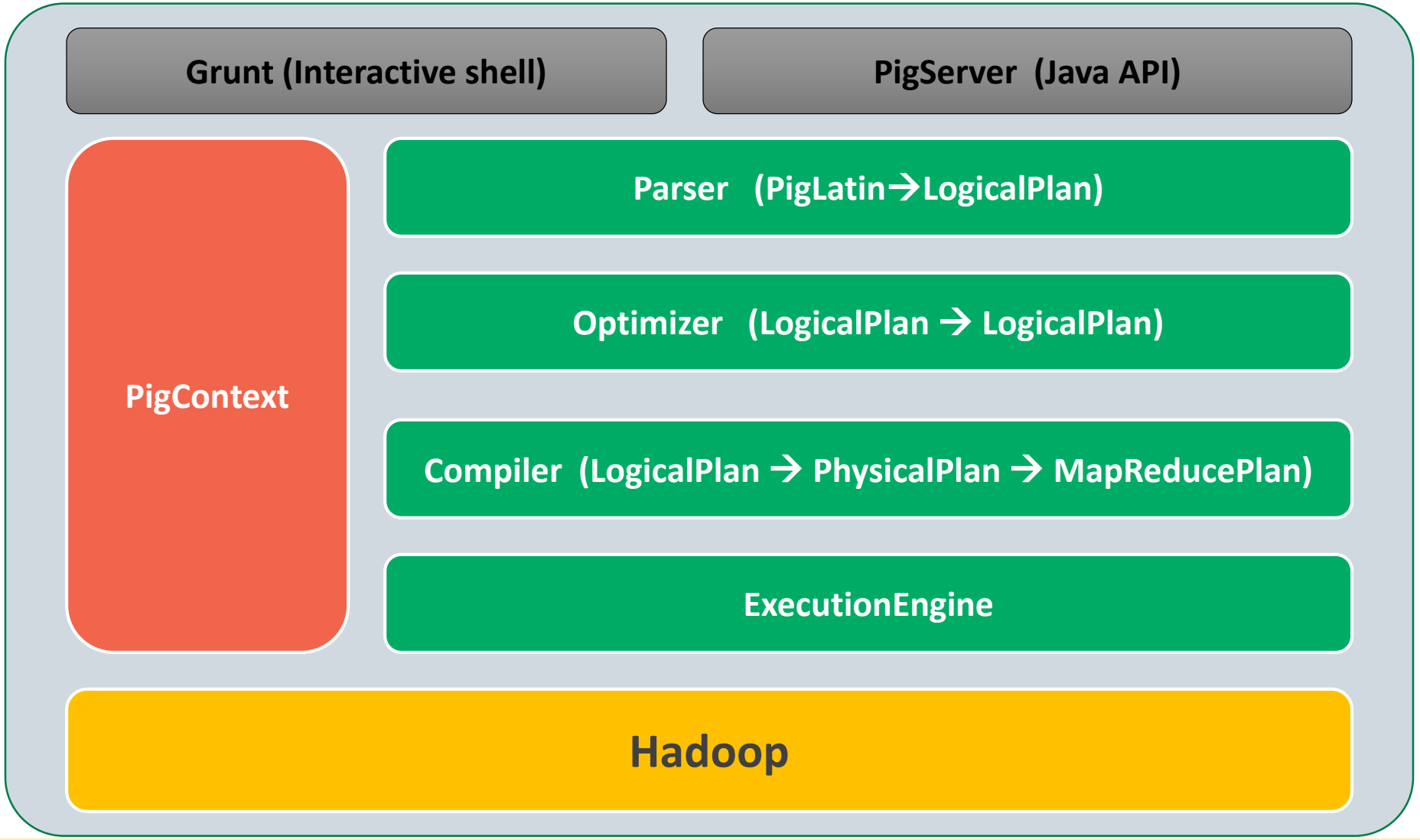
Pig Latin

Pig Latin a dataflow language is similar to SQL and it easy write a script if you are good at SQL.

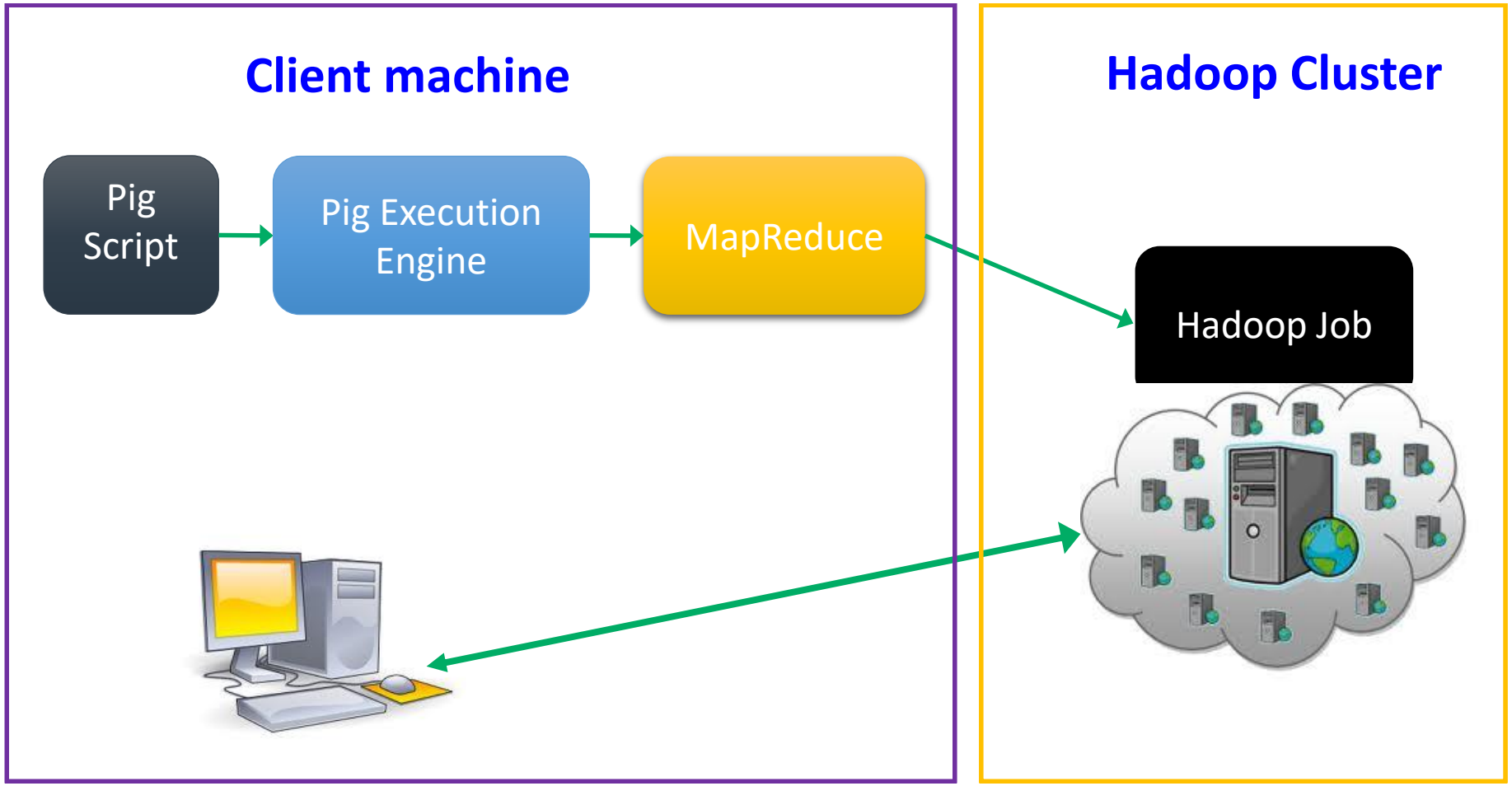
Pig Engine

Accepts the Pig Latin scripts as input and converts those into MapReduce jobs.

Pig High Level Architecture



Running Pig Latin Scripts



Data Analysis Example

We want to find top 5 most visited pages by users between 18-25

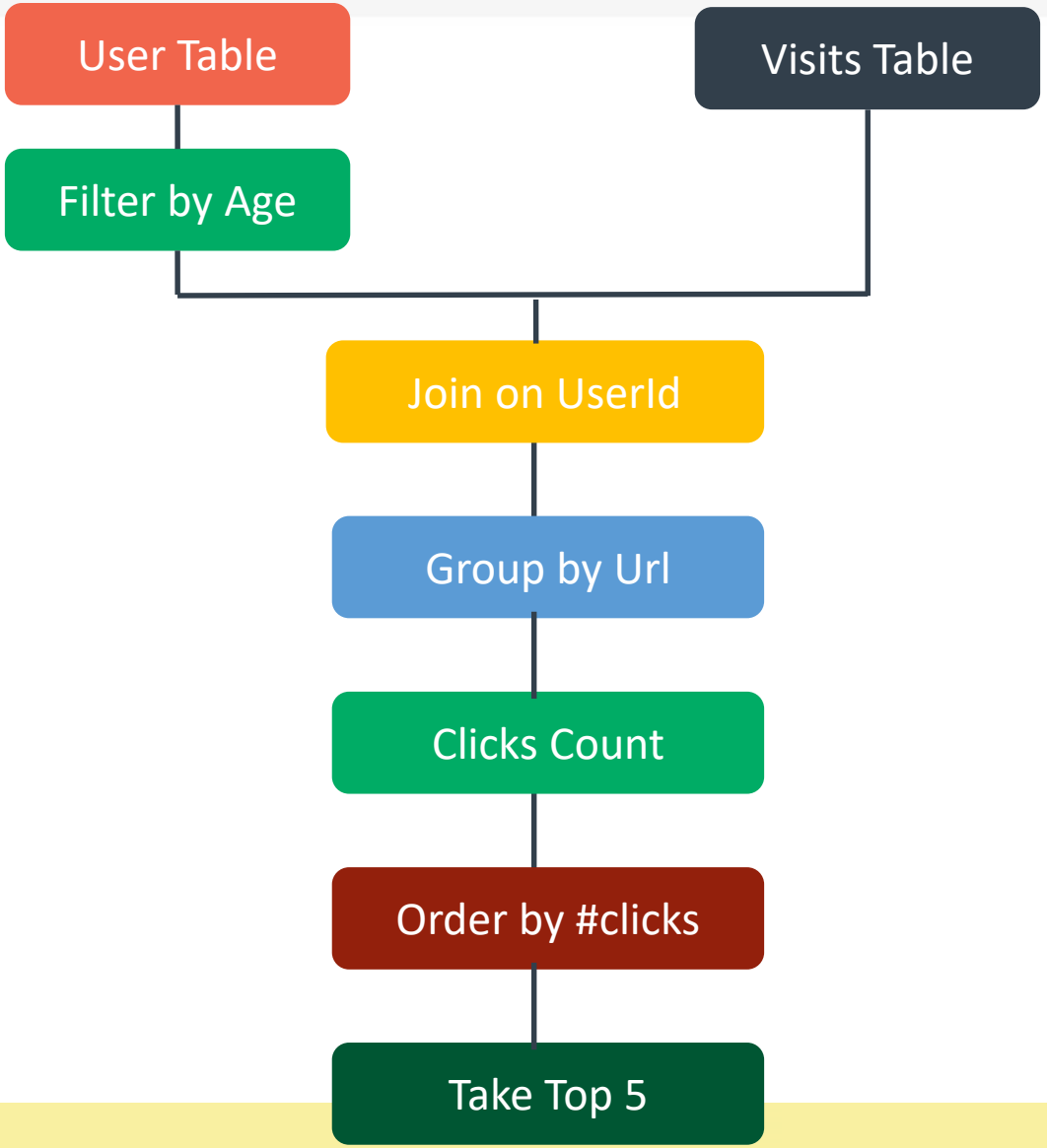
Visits

UserID	Url	Page	#Clicks	Time
123	cnn.com	home	2	8:00
123	bbc.com	contact	1	10:00
123	flickr.com	profile	3	10:05
654	cnn.com	services	1	12:00

Users

UserID	City	Age
123	New York	19
654	Boston	28
963	Los Angeles	17
789	Austin	21

Data Analysis Example



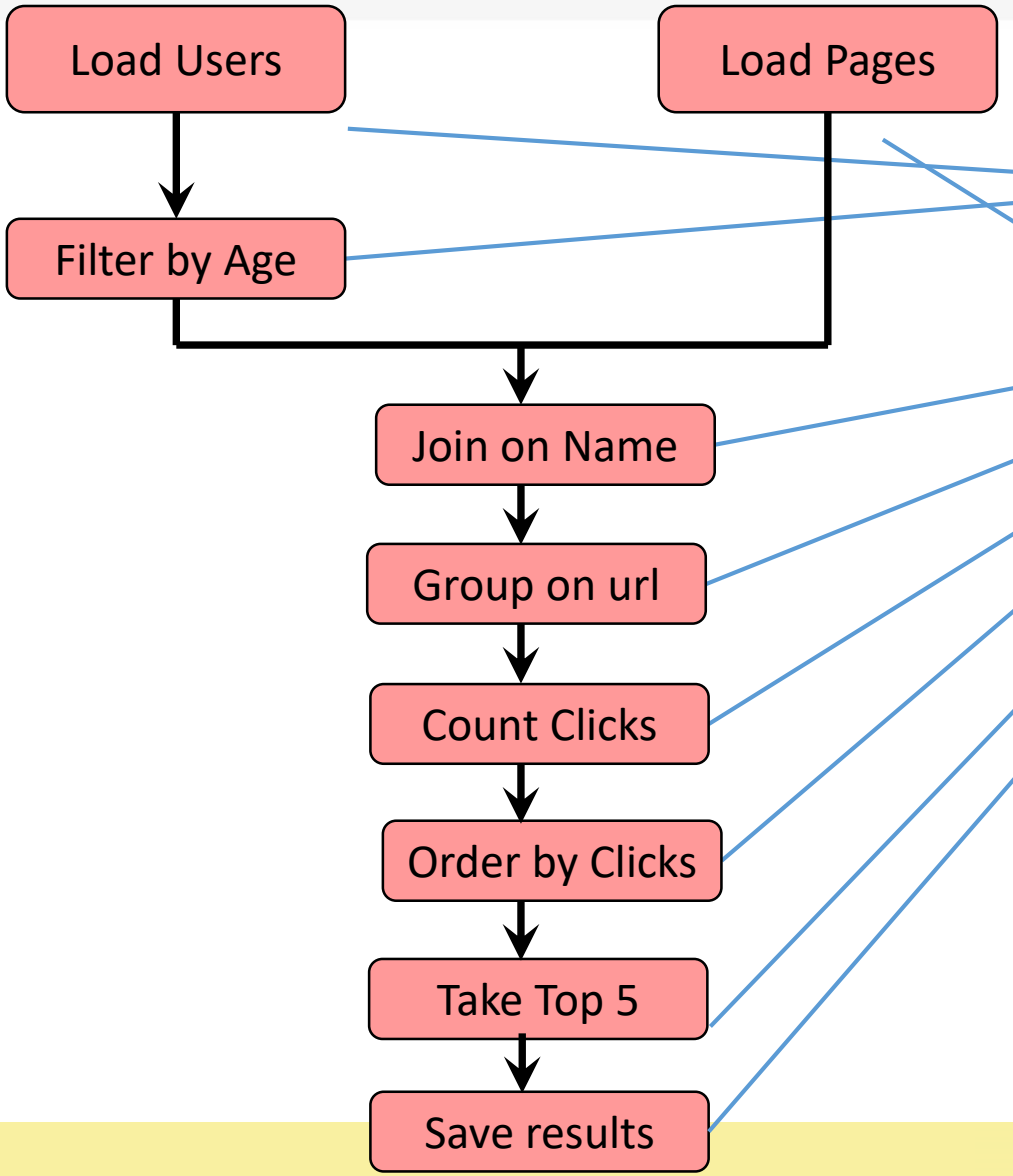
MapReduce (Java) = 200++ lines – 4 h

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException { }
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
            // store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to
                if (value.charAt(0) == '1')
                    first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableC
        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable
                Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }
            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, Text
        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter
        oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable,
        int count = 0;
        public void reduce(
                LongWritable key,
                Iterator<Text> iter,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
                Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
                new Path("/user/gates/tmp/indexed_p
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.cla
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.clas
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.c
        FileInputFormat.addInputPath(lfu, new
                Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
                new Path("/user/gates/tmp/filtered_
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFo
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
                Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
                Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
                Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputF
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.
        group.setOutputFormat(
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class
        group.setReducerClass(ReduceUrls.class)
        FileInputFormat.addInputPath(group, new
                Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, nev
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInput
        top100.setOutputKeyClass(LongWritable.c
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(
        top100.setMapperClass(LoadClicks.class)
        top100.setCombinerClass(LimitClicks.cla
        top100.setReducerClass(LimitClicks.class
        FileInputFormat.addInputPath(top100, ne
        FileOutputFormat.setOutputPath(top100, ne
                Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobCo
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Pig Latin = 10 lines – 10 mn

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

Pig Latin




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

pages = load 'pages.csv' as (

joined = join users_1825 by (
grouped = group joined by url
summed = foreach grouped gene
sorted = order summed by view
top_5 = limit sorted 5;

store top_5 into 'top_5_sites
```

Going from Pig Latin to MapReduce

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:

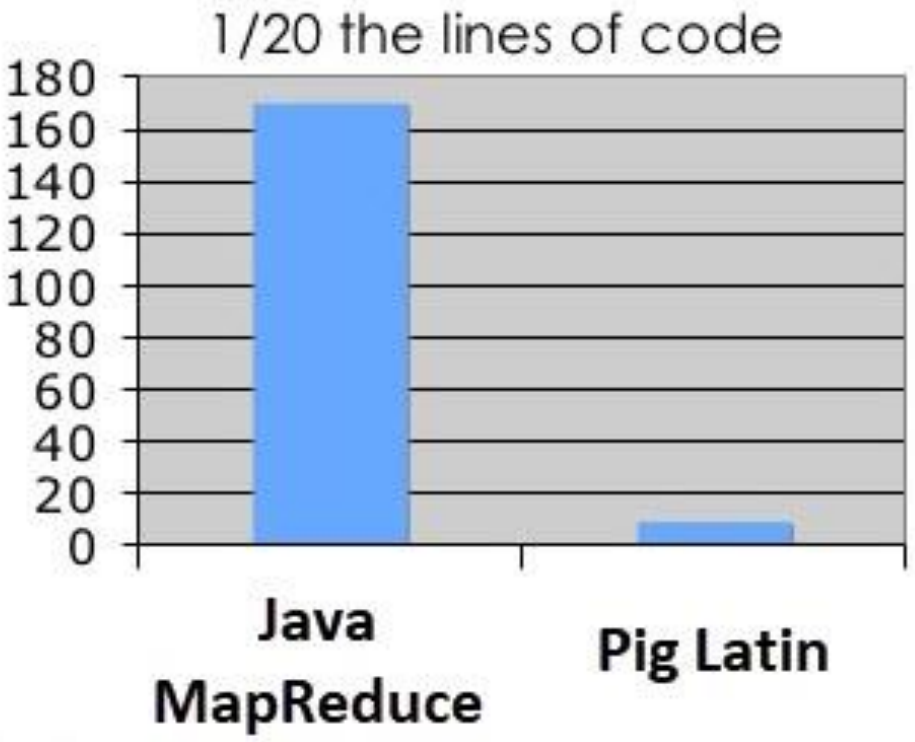
- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

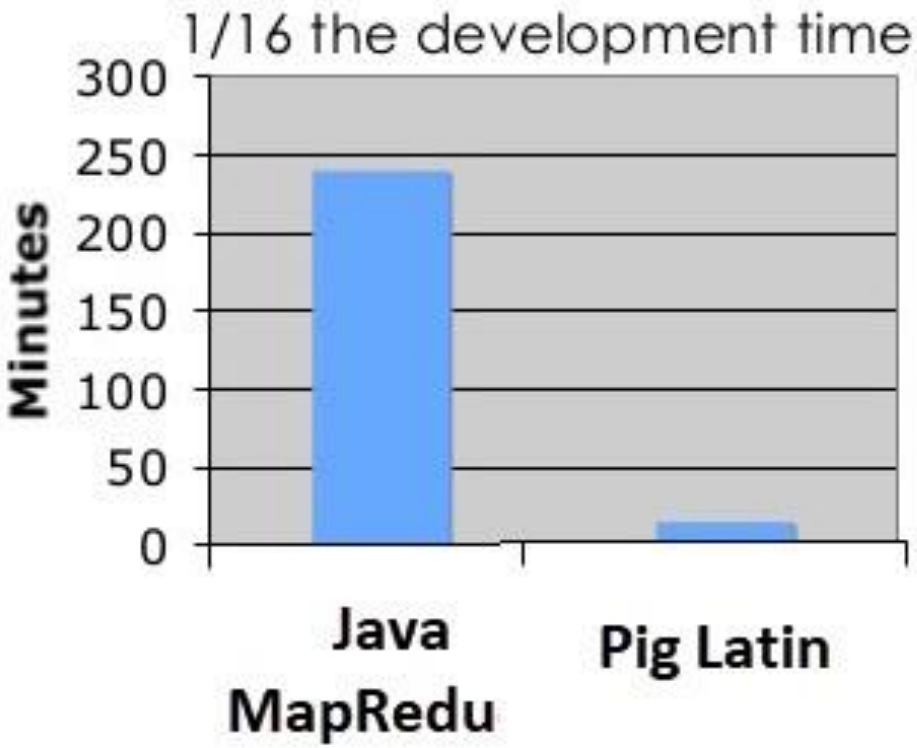
Execution Plan

Map:
Filter

Reduce:
Count

Apache Pig Performances





- 10 lines of PigLatin = 200+ lines in java
- 15 mn of PigLatin = 4 hours in java

Pig Execution Modes

- **Pig in Local mode**
  - No HDFS is required, All files run on local file system.
  - Command: `pig -x local`

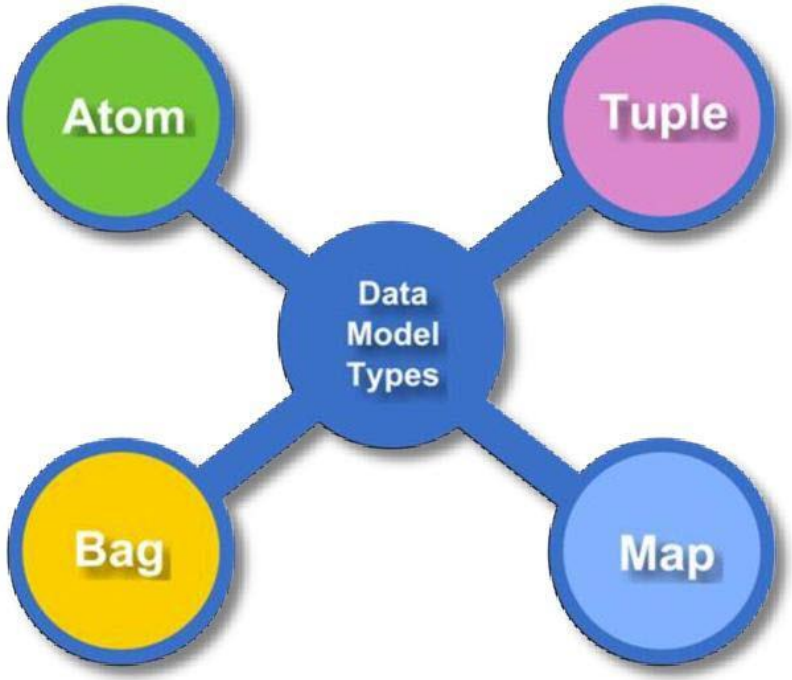
- **Pig in MapReduce (hadoop) mode – Default Mode**
  - To run PIG scripts in MR mode, ensure you have access to HDFS,
  - Command: `pig` or `pig -x mapreduce`

Introduction to Pig Latin

- Pig Latin is a high-level programming language
- It provides various operators for reading, writing, and processing data

- Pig Latin program
  - Describes a data flow
  - Made up of a series of operations (or transformations)
  - Each operation is applied to input data and produce output data

Pig Latin Data Model



Pig Latin four basics types

- **Atom** : Simple atomic value

  e.g: int, long, double, chararray, etc.

- **Tuple** : A sequence of field that can be any of the data types

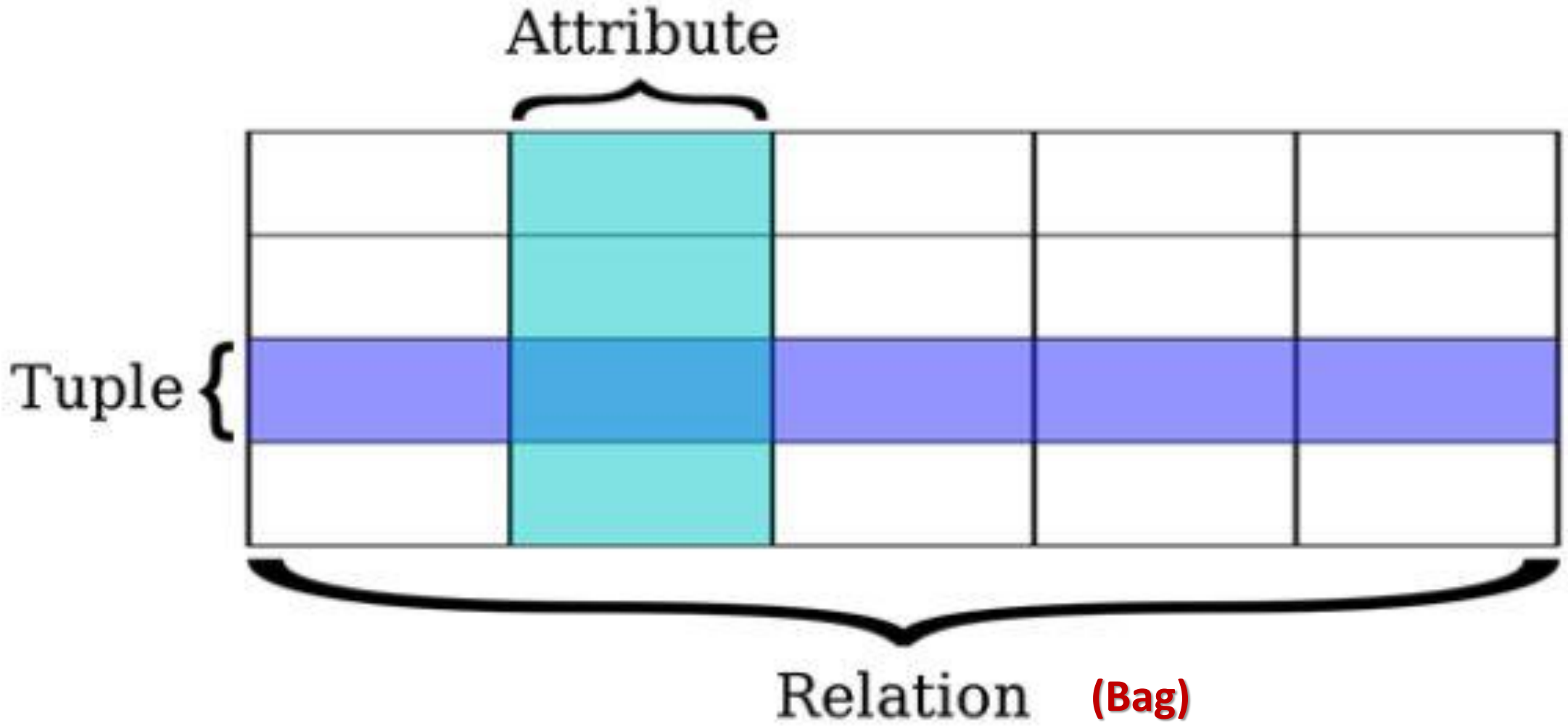
  e.g: ('AMY' ,19)

- **Bag** : A collection of Tuples of potentially varied structure. Can contains duplicates

  e.g: {('AMY'), ('John', (25,32))}

- **Map** : An associative array, the key must be a chararray, the value can be any type

Pig Latin Data Model



Pig Latin Statements

- Pig Latin statements are generally organized in the following manner:

  1. A **LOAD** statement reads data from the file system.
  2. A series of "transformation" statements process the data.
  3. A **STORE** statement writes output to the file system;

     OR
  4. A **DUMP** statement displays output to the screen

- The trigger for Pig to start execution are the **DUMP** and **STORE** statements

Pig Latin Statements – Operators (some)

Category	Operator	Description
Loading and Storing	LOAD STORE DUMP	Loads data from the file system or other storage into a relation . Saves a relation to the file system or other storage. Prints a relation to the console.
Filtering	FILTER DISTINCT FOREACH...GENERATE STREAM	Removes unwanted rows from a relation. Removes duplicate rows from a relation. Adds or removes fields from a relation. Transforms a relation using an external program.
Grouping and Joining	JOIN COGROUP GROUP CROSS	Joins two or more relations. Groups the data in two or more relations. Groups the data in a single relation. Creates the cross product of two or more relations.
Sorting	ORDER LIMIT	Sorts a relation by one or more fields. Limits the size of a relation to a maximum number of tuples.
Combining and Splitting	UNION SPLIT	Combines two or more relations into one. Splits a relation into two or more relations.

Pig Latin Script Example

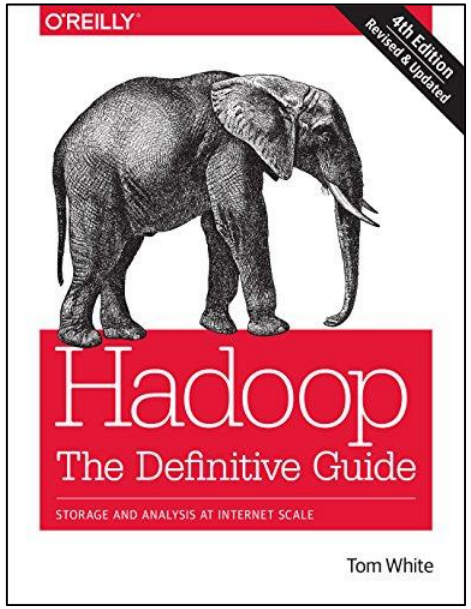
```
-- Load users and pages data files
Users = LOAD '/data/texts/users.txt' AS (user: chararray, age: int);
Pages = LOAD '/data/texts/pages.txt' AS (user: chararray, url: chararray);
-- Remain records with users with age between 18 and 25
Fltrd = FILTER Users BY age >= 18 and age <= 25;
-- Join data sets by a user key
Jnd = JOIN Fltrd BY user, Pages BY user;
-- Group records together by each url
Grpd = GROUP Jnd BY url;
-- Calculate click count for each group
Smmd = FOREACH Grpd GENERATE group, COUNT(Jnd) AS clicks;
-- Sort records by a numer of click
Srtd = ORDER Smmd BY clicks DESC;
-- Get top 5 pages
Lmt = LIMIT Srtd 5;
-- Store output records in a given directory
STORE Lmt INTO '/jobs/output/top5Pages';
```

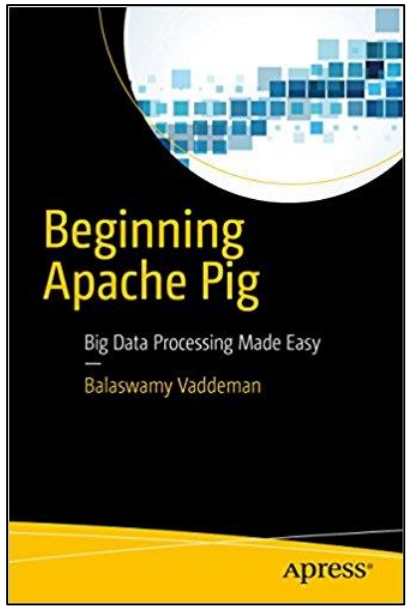
Pig Workshop

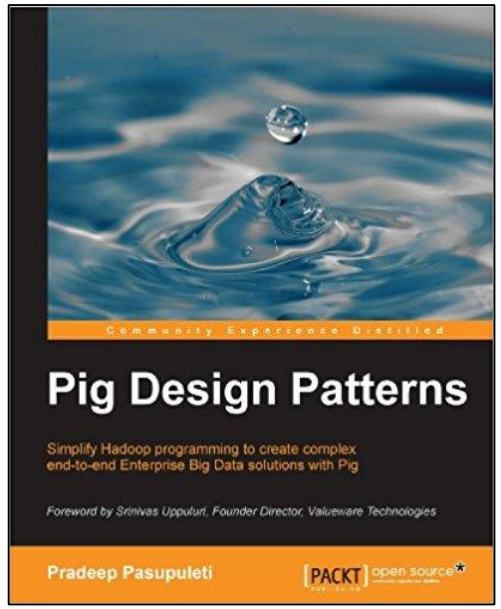
It’s time for a break

Grab some coffee, We’ll be back in 15min

Resources







Online Resources

https://pig.apache.org/

http://pig.apache.org/docs/r0.17.0/

Thank You