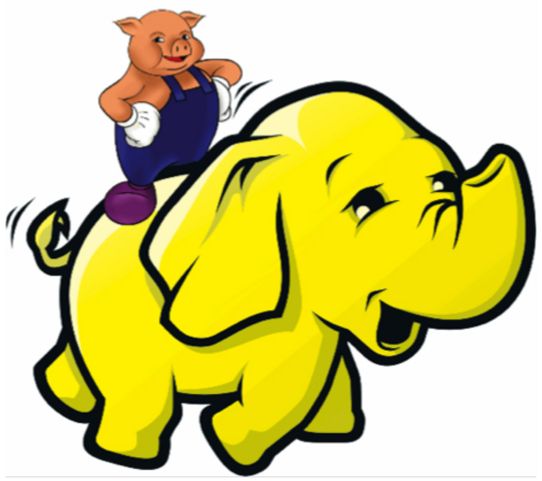
Apache Pig

Pig Latin

What is Pig

- An engine for executing programs on top of Hadoop
- It provides a language (called Pig Latin) for writing these programs
- It is an Apache open source project

https://pig.apache.org/



MapReduce

- Process is moved to the data
- A simple programming model
    - Map: each map task works on a key-value pair
    - shuffle & sort: group values by key
    - Reduce: each reduce task works on a key and has an iterator over all values associated with that key
- User provides: input, output, map function, reduce function
    - Can also control: partitioning, sorting, use combiner

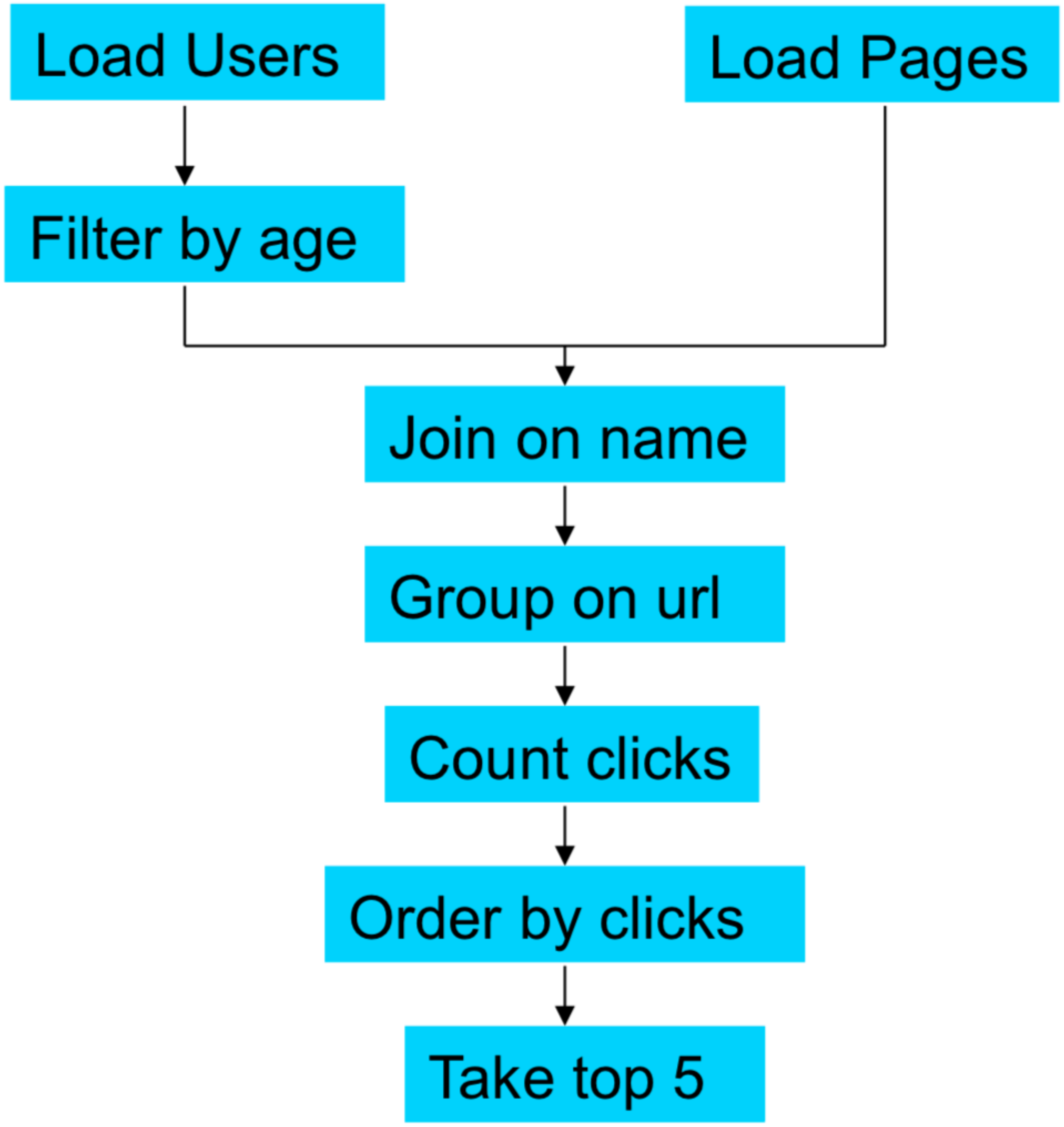
MapReduce is not Enough

- MapReduce is good for batch processing
  - not good for real time access
- for some complex processing, might require writing large amount of code, and write multiple MapReduce jobs
  - high-level language for access is needed

Why to use PigLatin

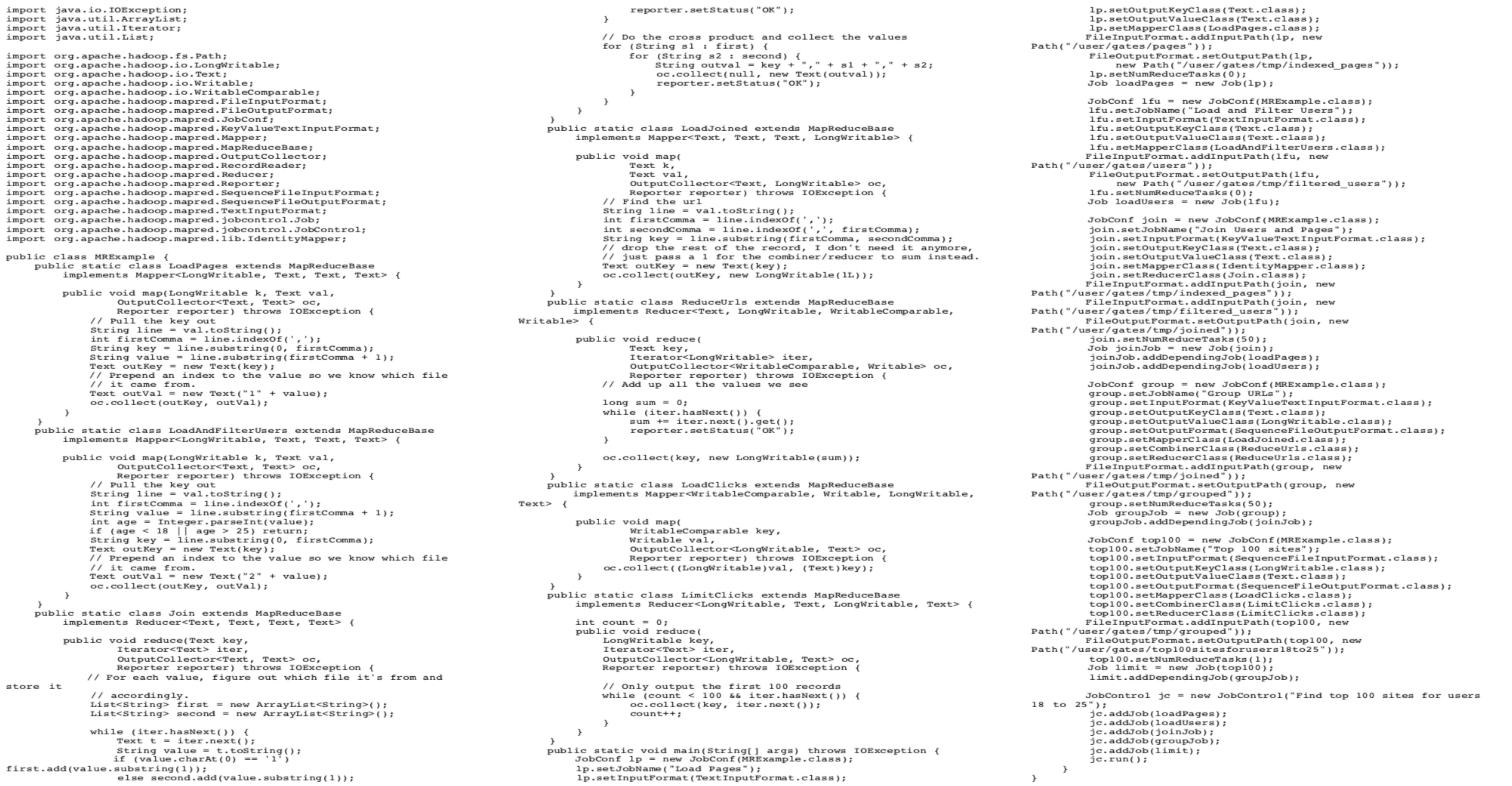
- Suppose we have two relations;
  - the first relation contains user data such as name and age
  - the second relation contains websites data; website, user(name), number of clicks
- we want to find the top 5 most visited websites by users aged between 18 & 25

Workflow



Load Users
Load Pages
Filter by age
Join on name
Group on url
Count clicks
Order by clicks
Take top 5

In MapReduce



```
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.JobControl;
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.toString();
                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.indexOf(',', firstComma);
            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, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                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, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        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_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

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

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        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 JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

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

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Many lines of code, takes long time to write

In Pig Latin

```
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’;
```

few lines of code, takes short time to write

Pig Latin

- Pig Latin is high-level language
  - a level higher than MapReduce
  - behind the scene; Pig latin statements are translated into MapReduce jobs by Pig engine
  - Provides common operations such as join, group, sort, filter, …
  - Schema is optional, can be defined at runtime
  - A great support for User Defined Functions (UDF)
- Is considered a data flow language
  - it shows the users the flow of what is happening in order

What are people doing with Pig Latin

- At Yahoo ~70% of Hadoop jobs are Pig jobs
- Used by big companies; Google, LinkedIn, Twitter, and Facebook
- Example of Pig task
  - Web log processing
  - Image Processing
  - …

Easy & Efficient

- Easy to write compared to writing MapReduce jobs
- Increases efficiency
    - In study test, they found that
        - 10 lines of Pig Latin ~ 200 lines of MapReduce Java code
        - what took 4 hours in Java to write a MapReduce took 15 minutes in Pig Latin
        - does not require Java programming skill

Pig Execution Mode

- Local mode
  - runs locally, no need for Hadoop MapReduce or HDFS
  - this mode is good for testing locally
- MapReduce mode
  - requires to have a Hadoop cluster
  - Pig Latin will read input data from HDFS
  - Pig script will be converted into MapReduce jobs

Start Pig from the Command Line

- download & install Pig from https://pig.apache.org/releases.html
- from the command line, you can choose the mode
  - local mode

    ```
    $ pig -x local  —> output is grunt>
    ```

  - mapreduce mode

    ```
    $ pig -x  mapreduce (or simply pig)
    ```

Execution Mechanism

- Interactive mode (Grunt shell)
    - run the Pig Latin statements in interactive mode, and see the dump of output after each statement
- Batch mode (script)
    - adding all related statements in one script and then run the script
    - script file extension is *.pig*

Types in PigLatin

- Atomic: String, int, float, double, chararray, bytearray
  - for example, “Ali” , 45
- Tuple: multiple fields of different values & types
  - (“Ali”, 45, “manager”)
- Bag: group of tuples
  - {(“Ali”, 45, “manager”), (“Sami”, 35, ”senior developer), …}
- Map: key, value
  - value can be single value, tuple, bag

Pig Latin Operators

Load

- The load statement is used to load input from specified relation
- Syntax: load ‘data’ [USING function] [AS schema]
  - data can be single file, if a directory is given, then all files will be loaded
  - USING is a keyword
  - function is the load function; how the data is read. we can use one of Pig built-in functions or provide our own function if the data is in format that can’t be processed by built-in functions
  - AS is a keyword
  - List of attributes (schema)
    - written inside parentheses

Load Example

- Example:
  - this will load content from file “query_log.txt” into relation queries

```
queries = LOAD ‘query_log.txt’ AS (userID, queryString, timeStamp)
```

  - by default the parser will assume that the input is tab separated
    - the first value is stored in userID
    - second in queryString
    - third is timeStamp
  - if the file contains more than three columns, the rest will be ignored

Load Examples

- A = load ‘myFile.txt’ ;
- A = load ‘myFile.txt’ USING PigStorage (`\t`);
- These two statements are equivalent USING PigStorage (`\t`) is the default
- in both the schema is not defined
  - all the fields will be of type bytearray
- schema can be specified using AS keyword
- A = load ‘myFile.txt’ AS (f1 : int , f2: String);
- To verify that load statement worked, use one of the diagnostic operators

Diagnostic Operators

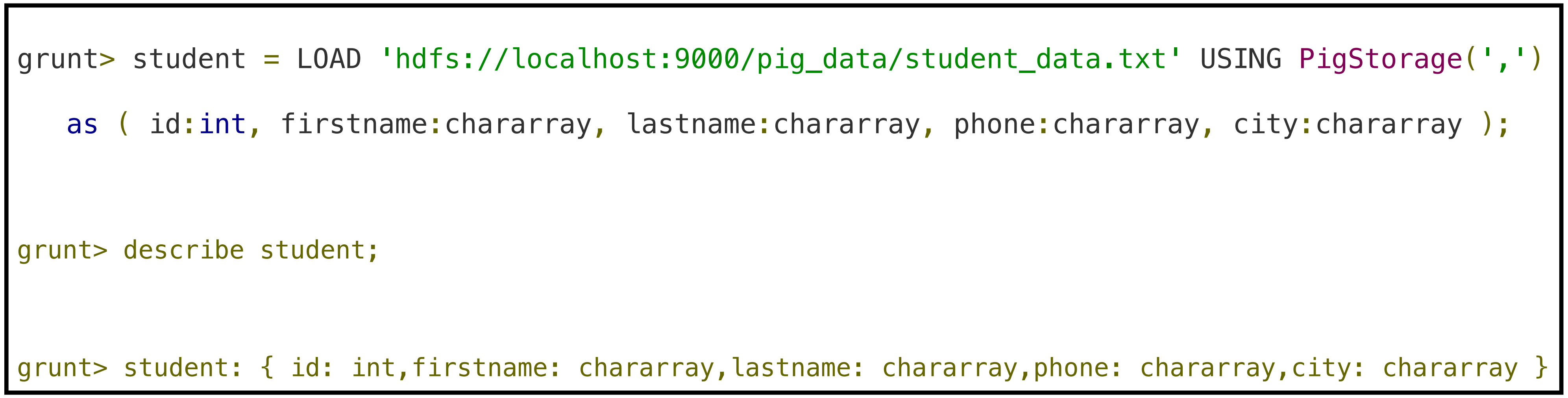
- Pig provides four different ways to diagnose the result of running a Pig Latin statement
    - dump operator
    - describe operator
    - explanation operator
    - illustration operator

Dump Operator

- is used to run the statement and display snippet of the result on the screen
- for example
    - > A = load ‘myFile.txt’;
    - > dump A;

Describe Operator

- is used to view the schema of the relation



```
grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student_data.txt' USING PigStorage(',')

   as ( id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray );


grunt> describe student;


grunt> student: { id: int,firstname: chararray,lastname: chararray,phone: chararray,city: chararray }
```

Explain & illustrate Operator

- is used to display the execution plan, MapReduce jobs plan
- takes sample of the input data and show how the job will run
- grunt > explain customers;