Big Data
- Data Flow with Pig

Dr. Qing “Matt” Zhang
ITU

Outline

- Introduction
- Pig Latin scripts
- Pig Data types and operators
- Optimizations

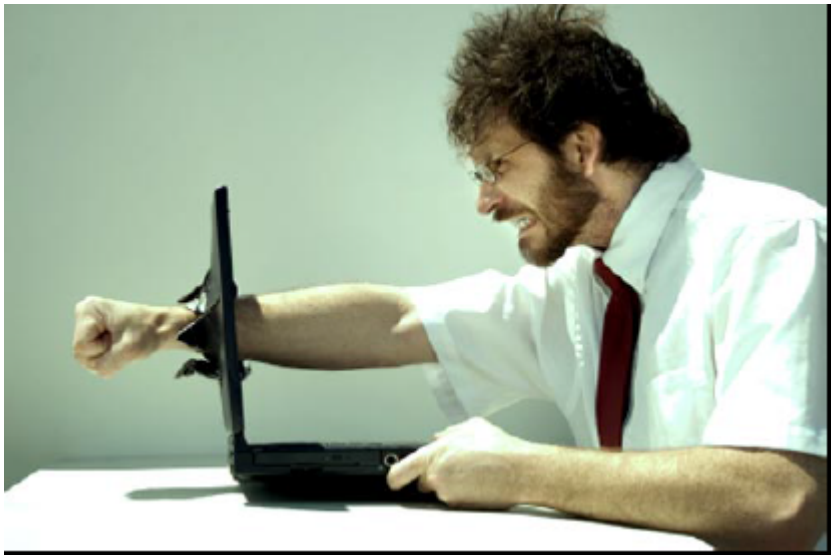
Introduction

What can I tell about MapReduce?

- Developing MapReduce jobs can be challenging

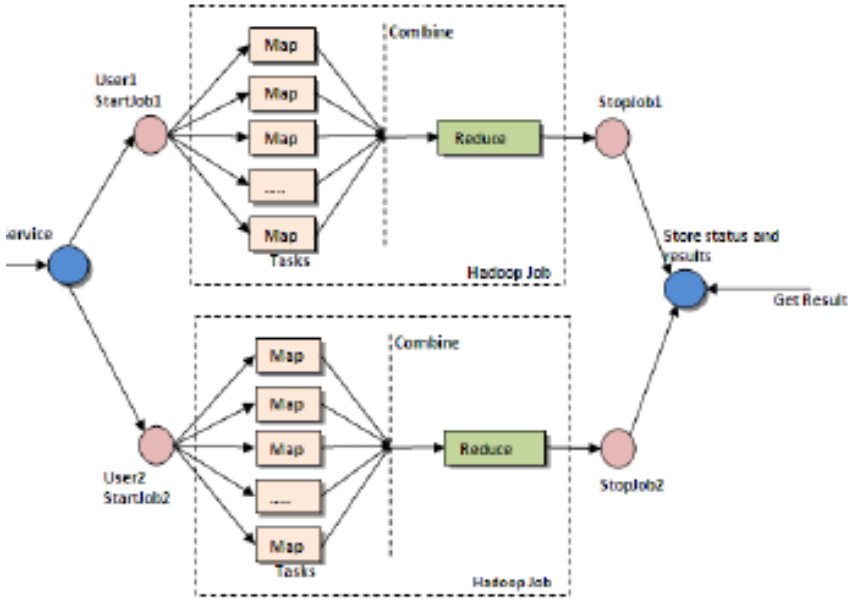
What can I tell about MapReduce?

- Developing MapReduce jobs can be challenging
  - Jobs are written (mainly) in Java



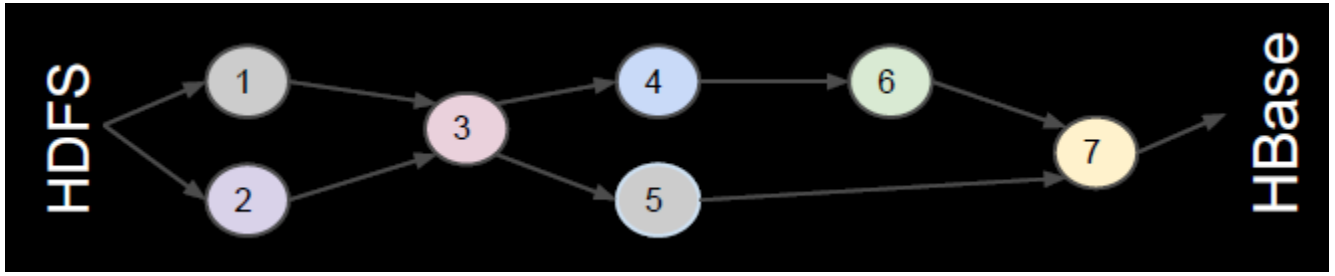
What can I tell about MapReduce?

- Developing MapReduce jobs can be challenging
  - Jobs are written (mainly) in Java
  - Must think in terms of map and reduce



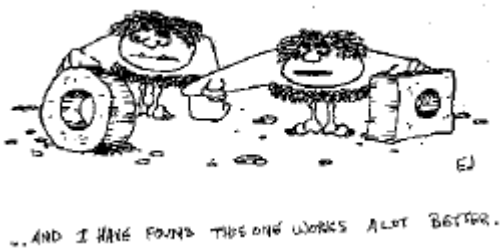
What can I tell about MapReduce?

- Developing MapReduce jobs can be challenging
  - Jobs are written (mainly) in Java
  - Must think in terms of map and reduce
  - n-stage jobs can be difficult to manage



What can I tell about MapReduce?

- Developing MapReduce jobs can be challenging
  - Jobs are written (mainly) in Java
  - Must think in terms of map and reduce
  - n-stage jobs can be difficult to manage
  - Many common operations (e.g. filters, projections, joins) requires a custom code



What can I tell about MapReduce?

- Developing MapReduce jobs can be challenging
  - Jobs are written (mainly) in Java
  - Must think in terms of map and reduce
  - n-stage jobs can be difficult to manage
  - Many common operations (e.g. filters, projections, joins) requires a custom code
  - Rich data types also require a custom code

WordCount in Java

Source Code

WordCount.java

```
package org.myorg;

import java.io.*;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount extends Configured implements Tool {

 public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

  static enum Counters { INPUT_WORDS }

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  private boolean caseSensitive = true;
  private Set<String> patternsToSkip = new HashSet<String>();

  private long numRecords = 0;
  private String inputFile;

  public void configure(JobConf job) {
   caseSensitive = job.getBoolean("wordcount.case.sensitive", true);
   inputFile = job.get("map.input.file");

   if (job.getBoolean("wordcount.skip.patterns", false)) {
    Path[] patternsFiles = new Path[0];
    try {
     patternsFiles = DistributedCache.getLocalCacheFiles(job);
    } catch (IOException ioe) {
     System.err.println("Caught exception while getting cached files: " + StringUtils.stringifyException(ioe));
    }
    for (Path patternsFile : patternsFiles) {
     parseSkipFile(patternsFile);
    }
   }
  }

  private void parseSkipFile(Path patternsFile) {
   try {
    BufferedReader fis = new BufferedReader(new FileReader(patternsFile.toString()));
    String pattern = null;
    while ((pattern = fis.readLine()) != null) {
     patternsToSkip.add(pattern);
    }
   } catch (IOException ioe) {
    System.err.println("Caught exception while parsing the cached file '" + patternsFile + "' : " +
StringUtils.stringifyException(ioe));
   }
  }

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
   String line = (caseSensitive) ? value.toString() : value.toString().toLowerCase();

   for (String pattern : patternsToSkip) {
    line = line.replaceAll(pattern, "");
   }

   StringTokenizer tokenizer = new StringTokenizer(line);
   while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
    reporter.incrCounter(Counters.INPUT_WORDS, 1);
   }

   if ((++numRecords % 100) == 0) {
    reporter.setStatus("Finished processing " + numRecords + " records " + "from the input file: " + inputFile);
   }
  }
 }

 public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
   int sum = 0;
   while (values.hasNext()) {
    sum += values.next().get();
   }
   output.collect(key, new IntWritable(sum));
  }
 }

 public int run(String[] args) throws Exception {
  JobConf conf = new JobConf(getConf(), WordCount.class);
  conf.setJobName("wordcount");

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

  List<String> other_args = new ArrayList<String>();
  for (int i=0; i < args.length; ++i) {
   if ("-skip".equals(args[i])) {
    DistributedCache.addCacheFile(new Path(args[++i]).toUri(), conf);
    conf.setBoolean("wordcount.skip.patterns", true);
   } else {
    other_args.add(args[i]);
   }
  }

  FileInputFormat.setInputPaths(conf, new Path(other_args.get(0)));
  FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));

  JobClient.runJob(conf);
  return 0;
 }

 public static void main(String[] args) throws Exception {
  int res = ToolRunner.run(new Configuration(), new WordCount(), args);
  System.exit(res);
 }
}
```

What is Pig?

Platform for easier analyzing large data sets

- PigLatin
  - Simple but powerful data flow language similar to scripting languages
- Pig Engine
  - Parses, optimizes and automatically executes PigLatin scripts as series of MapReduce jobs on Hadoop cluster

Pig Concepts

- In Pig, a single element of data is an *atom*
  - Analogous to a field
- A collection of atoms is a *tuple*
  - Analogous to a row, or partial row, in database terminology
- A collection of tuples is a *bag*
  - Sometimes known as a relation or result set
- Typically, a PigLatin script starts by loading one or more datasets into bags, and then creates new bags by modifying those it already has

What does PigLatin offer?

PigLatin is a high level and easy to understand data flow programming language

- It provides
  - common data operations
    - filters
    - joins
    - ordering
  - nested types
    - tuples, bags, maps

What does PigLatin offer?

PigLatin is a high level and easy to understand data flow programming language

- More natural for analysts than MapReduce
- Opens Hadoop to non-Java programmers

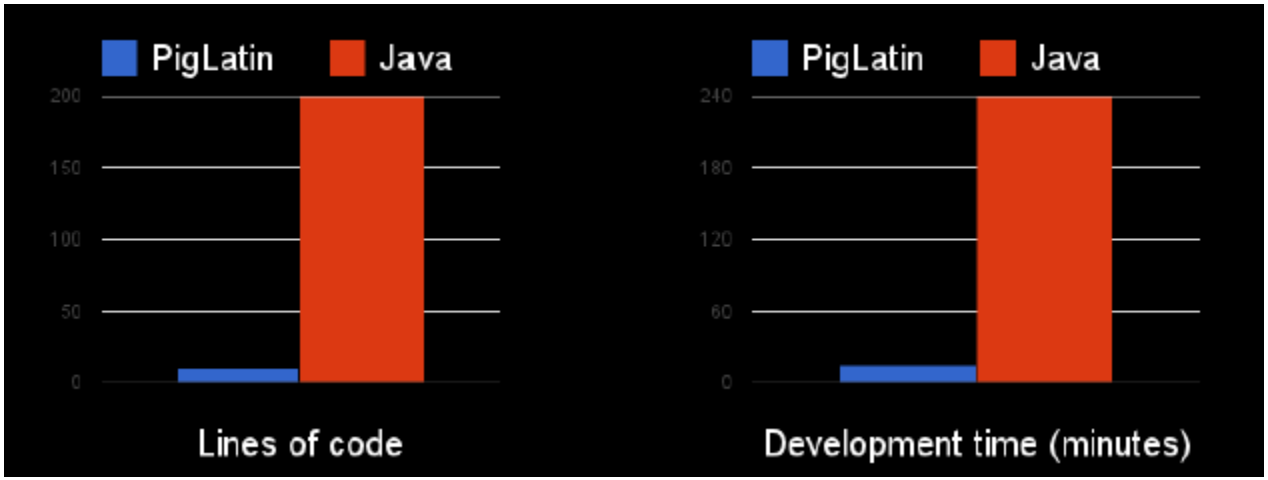
How does WordCount in PigLatin look like?

popular.pig | wordcount.pig

```
Lines = LOAD '/data/texts/*.txt' AS (line:chararray);
-- TOKENIZE splits the line into a bag of words
-- FLATTEN produces a separate record for each item
--    from a bag
Words = FOREACH Lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
-- Group records together by each word.
Groups = GROUP Words BY word;
-- Count words.
Counts = FOREACH Groups GENERATE group, COUNT(Words);
-- Store the results.
STORE Counts INTO 'counts';
```

What is the biggest benefit of Pig?

- Dramatically increases productivity
- One test: **Find the top 5 most visited pages by users aged 18-25**
- 10 lines of PigLatin = 200 lines in Java
- 15 minutes in PigLatin = 4 hours in Java



How to find such pages in PigLatin?

```
-- 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';
```

What are other Pig Benefits?

- Manages all details of submitting jobs and running complex data flows
- Insulates from changes in Hadoop Java API

- Easy extensible by UDFs
- Embeddable in Python and JavaScript
- Integrated with HBase
- Actively supported by the community

How is Pig Being Used

- Rapid prototyping of algorithms for processing large data sets
- Data processing for web search platforms
- Ad hoc queries across large data sets
- Web log processing

Hive vs Pig

	Hive	Pig
Language	HiveQL (SQL-like)	Pig Latin, a data flow language
Schema	Table definitions that are stored in a metastore	A schema is optionally defined at runtime. Metastore coming soon
Programmatic access	JDBC	PigServer (Java API)

Similarities

- Standard features such as filtering data, joining data sets, grouping and ordering
- Extensibility: Java UDFs and custom scripts
- Custom input and output formats
- Client-side shell access

Choose between Pig and Hive

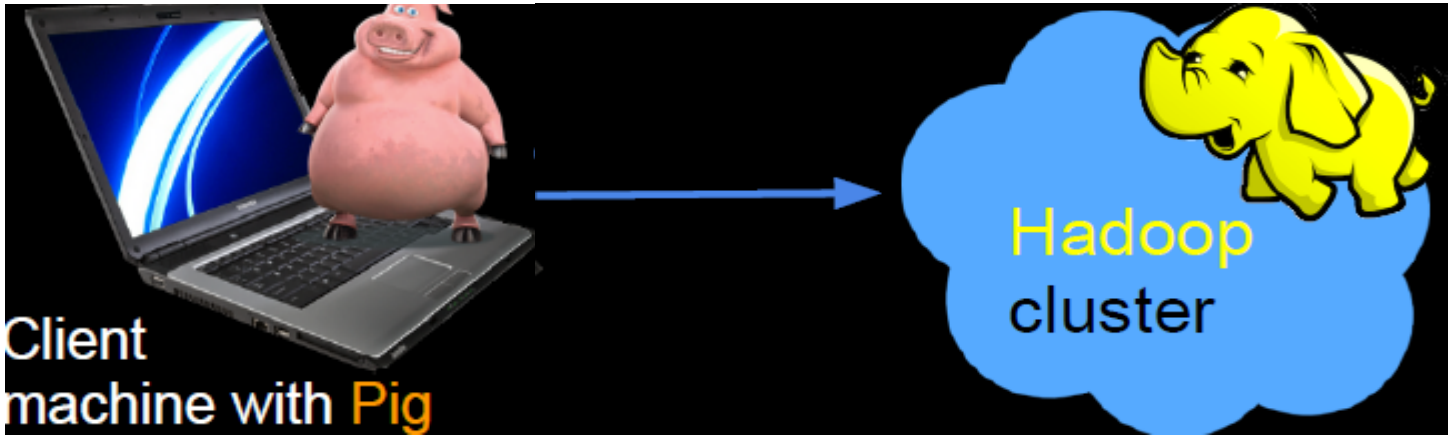
- Typically, organizations wanting an abstraction on top of standard MapReduce will choose to use either Hive or Pig
  - Which one is chosen depends on the skillset of the target users
    - Those with an SQL background will naturally gravitate towards Hive
    - Those who do not know SQL will often choose Pig
  - Hive uses SQL, but you lose some ability to optimize the query, by relying on the Hive optimizer.
  - Pig requires more verbose coding, but gives you more control and optimization over the flow of the data than Hive does

Choose between Pig and Hive

- Some organizations are now choosing to use both
  - Pig deals better with less-structured data,
    - used to manipulate the data into a more structured form, then Hive is used to query that structured data
  - Use Hive for some simple ad-hoc analytical queries into the data in HDFS
  - Use Pig to construct data flows, and run/tune automated batch jobs

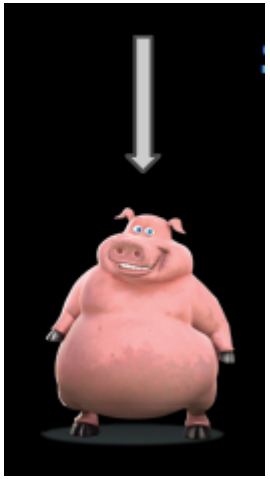
Where does Pig live?

- Pig is a client-side application, installed on user machine
- No need to install anything on the Hadoop cluster
- Pig and Hadoop versions must be compatible
- Pig submits and executes jobs to the Hadoop cluster



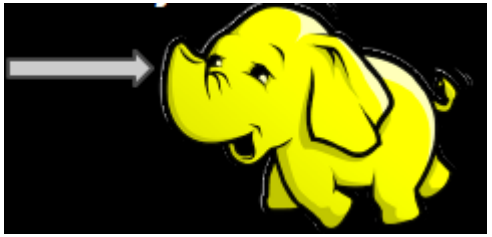
How does Pig Work?

```
Lines = LOAD '/data/texts/*.txt' AS (line:chararray);
Words = FOREACH Lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE group, COUNT(Words);
STORE Counts INTO 'counts';
```



${PIG_HOME}/bin/pig wordcount.pig

- parses
- optimizes
- plans execution
  - here as one MapReduce job
- submits jar to Hadoop
- monitors job progres

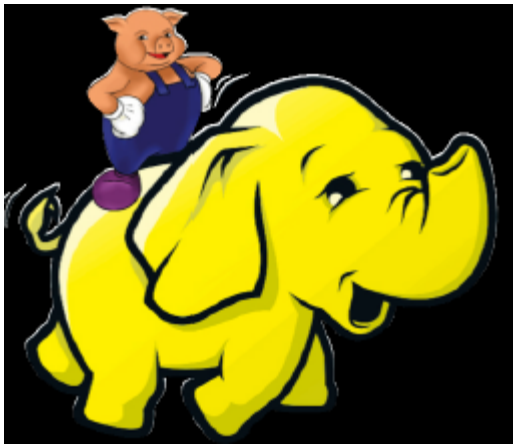


How to develop PigLatin Scripts?

- Eclipse plugins
  - PigEditor
    - syntax/errors highlighting
    - check of alias name existence
    - auto completion of keywords, UDF names
  - Pig-Eclipse
- Plugins for Vim, Emacs, TextMate
  - Usually provide syntax highlighting and code completion

Pig's Modes

- Local mode
  - Neither Hadoop nor HDFS is required
  - Local host and local file system is used
  - Useful for prototyping and debugging
- Hadoop mode
  - Take advantage of a Hadoop cluster and HDFS



How to Run PigLatin Scripts?

- Run a script directly - batch mode

  $ *pig -p input=someInput script.pig*

  script.pig:

  Lines = LOAD '$input' AS (...);

- Grunt, the Pig shell - interactive mode

```
grunt> Lines = LOAD '/data/books/'
            AS (line: chararray);
grunt> Unique = DISTINCT Lines;
grunt> DUMP Unique;
```

How to Run PigLatin Scripts?

- PigServer Java class, a JDBC like interface
- Python and JavaScript with PigLatin code embedded
  - adds control flow constructs such as if and for
  - avoids the need to invent a new language
  - uses a JDBC-like compile, bind, run model

Where to find useful PigLatin scripts?

- PiggyBank - Pig's repository of user contributed functions
  - load/store functions (e.g. from XML)
  - datetime, text functions
  - math, stats functions
- DataFu - LinkedIn's collection of Pig UDFs
  - statistics functions (quantiles, variance etc.)
  - convenient bag functions (intersection, union etc.)
  - utility functions (assertions, random numbers, MD5, distance between lat/long pair), PageRank

Pig vs MapReduce

- Pig translates to MR, so it can not be faster than well implemented MR code
- Pig may be faster than someone's MR code thanks to lots of optimizations
- Beginner's MR code will often lose out to Pig

Basic Pig Tutorial

PigLatin Data Types

- Scalar types
  - int, long, float, double, chararray, bytearray, boolean
- Complex types
  - tuple - sequence of fields of any type
    - ('Poland', 2, 0.66)
  - bag - an unordered collection of tuples, possibly with duplicates
    - {('Poland', 2), ('Greece'), (3.14)}

PigLatin Data Types

- Complex types
  - map - a set of key-value pairs. Keys must be a chararray, values may be any type
  - ['Poland'#'Euro2012']

PigLatin

- Pig relation is a bag of tuples
- Pig Latin statements work with relations

```
Users = LOAD 'users.txt' AS (user: chararray, age: int);
Fltrd = FILTER Users BY age >= 18 and age <= 25;
Srtd = ORDER Fltrd BY age DESC;
Lmt = LIMIT Srtd 3;
-- comment, Lmt is {(H,25),(C,22),(E,22)}

DUMP Lmt;
(H,25)
(C,22)
(E,22)
```

Last.fm music recommendation dataset

Last.fm Dataset - 1K users

- Data contained in two tab-separated files
  - tracks listened to by nearly 1000 users
  - users profiles
- ~2.35GB of total size (uncompressed)
- Publicly available for download
  - http://bit.ly/bsVBK3

Last.fm music recommendation dataset

- userid-timestamp-artid-artname-traidtraname.tsv (renamed here to userid-track.tsv) (~2.35GB)

  userid <tab> **timestamp** <tab> **musicbrainz-artist-id** <tab> **artist-name** <tab> **musicbrainz-track-id** <tab> **track-name**

- userid-profile.tsv (~37KB)

  userid <tab> **gender ('m'|'f'|empty)** <tab> **age (int|empty)** <tab> **country (str|empty)** <tab> **signup (date|empty)**

Example 1
Find number of tracks per artists

```
--Load the input file
Songs = LOAD 'whug/data/lastfm-dataset-1K/userid-track.tsv'
        AS (userId, timestamp, mbArtistId,
        artistName, mbTrackId, trackName);

-- Remain only artist and track names
Projected = FOREACH Songs GENERATE artistName, trackName;

-- Find unique records
Unique = DISTINCT Projected;

-- Group records by artist name
Grouped = GROUP Unique BY artistName;
-- Count the number of songs per artist
Counted = FOREACH Grouped GENERATE group, COUNT(Unique) AS cnt;

-- Displays a step-by-step execution of a sequence of statements
ILLUSTRATE Counted;
```

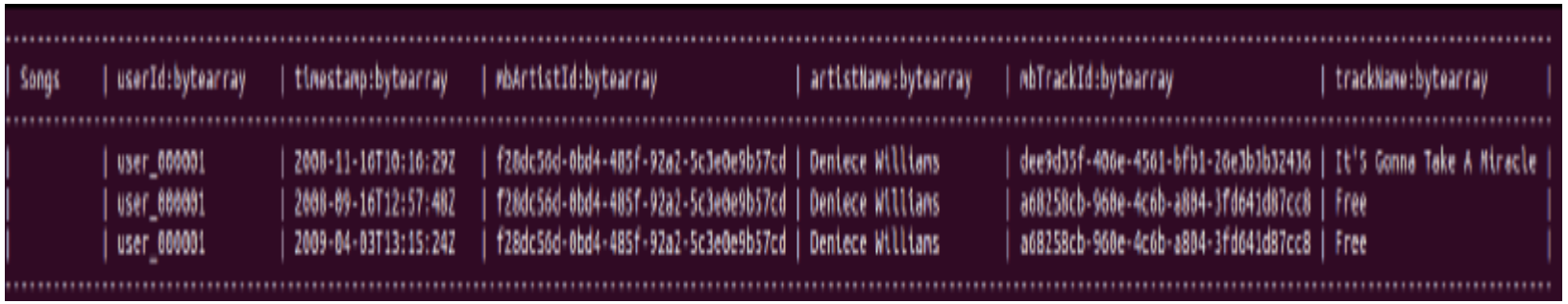
Example 1
Find number of tracks per artists

```
| Songs    | userId:bytearray | timestamp:bytearray  | mbArtistId:bytearray                 | artistName:bytearray | mbTrackId:bytearray                  | trackName:bytearray          |
|          | user_000001      | 2008-11-16T10:16:29Z | f28dc56d-0bd4-485f-92a2-5c3e0e9b57cd | Deniece Williams     | dee9d35f-406e-4561-bfb1-26e3b3b32436 | It'S Gonna Take A Miracle    |
|          | user_000001      | 2008-09-16T12:57:48Z | f28dc56d-0bd4-485f-92a2-5c3e0e9b57cd | Deniece Williams     | a68258cb-960e-4c6b-a804-3fd641d87cc8 | Free                         |
|          | user_000001      | 2009-04-03T13:15:24Z | f28dc56d-0bd4-485f-92a2-5c3e0e9b57cd | Deniece Williams     | a68258cb-960e-4c6b-a804-3fd641d87cc8 | Free                         |

| Projected | artistName:bytearray | trackName:bytearray          |
|           | Deniece Williams     | It'S Gonna Take A Miracle    |
|           | Deniece Williams     | Free                         |
|           | Deniece Williams     | Free                         |

| Unique | artistName:bytearray | trackName:bytearray          |
|        | Deniece Williams     | Free                         |
|        | Deniece Williams     | It'S Gonna Take A Miracle    |

| Grouped | group:bytearray | Unique:bag{:tuple(artistName:bytearray,trackName:bytearray)}                                  |
|         | Deniece Williams | {(Deniece Williams, Free), (Deniece Williams, It'S Gonna Take A Miracle)}                    |

| Counted | group:bytearray | cnt:long |
|         | Deniece Williams | 2        |
```

Pig Operations - Loading data

- LOAD loads input data

```
Songs = LOAD 'userid-track.tsv' AS
(userId, timestamp, mbArtistId,
artistName, mbTrackId, trackName);
```



Songs	userId:bytearray	timestamp:bytearray	mbArtistId:bytearray	artistName:bytearray	mbTrackId:bytearray	trackName:bytearray
	user_000001	2008-11-10T10:10:29Z	f28dc50d-0bd4-485f-92a2-5c3e0e9b57cd	Deniece Williams	dee9d35f-406e-4561-bfb1-26e3b1b32436	It'S Gonna Take A Miracle
	user_000001	2008-09-16T12:57:48Z	f28dc50d-0bd4-485f-92a2-5c3e0e9b57cd	Deniece Williams	a68258cb-960e-4c6b-a804-3fd641d87cc8	Free
	user_000001	2009-04-03T13:15:24Z	f28dc50d-0bd4-485f-92a2-5c3e0e9b57cd	Deniece Williams	a68258cb-960e-4c6b-a804-3fd641d87cc8	Free

Pig Operations - Loading data

- LOAD loads input data in various formats
- by default, loads data as tab-separated text files

```
Songs = LOAD 'lastfm/songs.tsv';
Songs = LOAD 'lastfm/*.csv' USING PigStorage(',');
```

- schema can be specified (defaults to bytearray)

```
Songs = LOAD 'lastfm/songs.tsv' AS (user: chararray, timestamp, artist, track: chararray);
```

Pig Operations - Loading data

- data from HBase table can be loaded

```
Songs = LOAD 'hbase://SongsTable' USING
HBaseStorage();
```

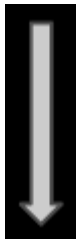
- a custom Load UDF can be used (see also PiggyBank)

```
Songs = LOAD 'lastfm' USING
MyCustomLoadUDF();
```

Pig Operations - Projection

- FOREACH ... GENERATE takes a set of expressions and applies them to every record

Songs	userId:bytearray	timestamp:bytearray	mbArtistId:bytearray	artistName:bytearray	mbTrackId:bytearray	trackName:bytearray
	user_000001	2008-11-16T10:10:29Z	f28dc50d-0bd4-485f-92a2-5c3e0e9b57cd	Deniece Williams	dee9d35f-406e-4501-bfb1-26e3b3b32436	It'S Gonna Take A Miracle
	user_000001	2008-09-16T12:57:48Z	f28dc50d-0bd4-485f-92a2-5c3e0e9b57cd	Deniece Williams	a68258cb-960e-4c6b-a804-3fd641d87cc8	Free
	user_000001	2009-04-03T13:15:24Z	f28dc50d-0bd4-485f-92a2-5c3e0e9b57cd	Deniece Williams	a68258cb-960e-4c6b-a804-3fd641d87cc8	Free



```
Projected = FOREACH Songs
        GENERATE artistName, trackName;
```

Projected	artistName:bytearray	trackName:bytearray
	Deniece Williams	It'S Gonna Take A Miracle
	Deniece Williams	Free
	Deniece Williams	Free

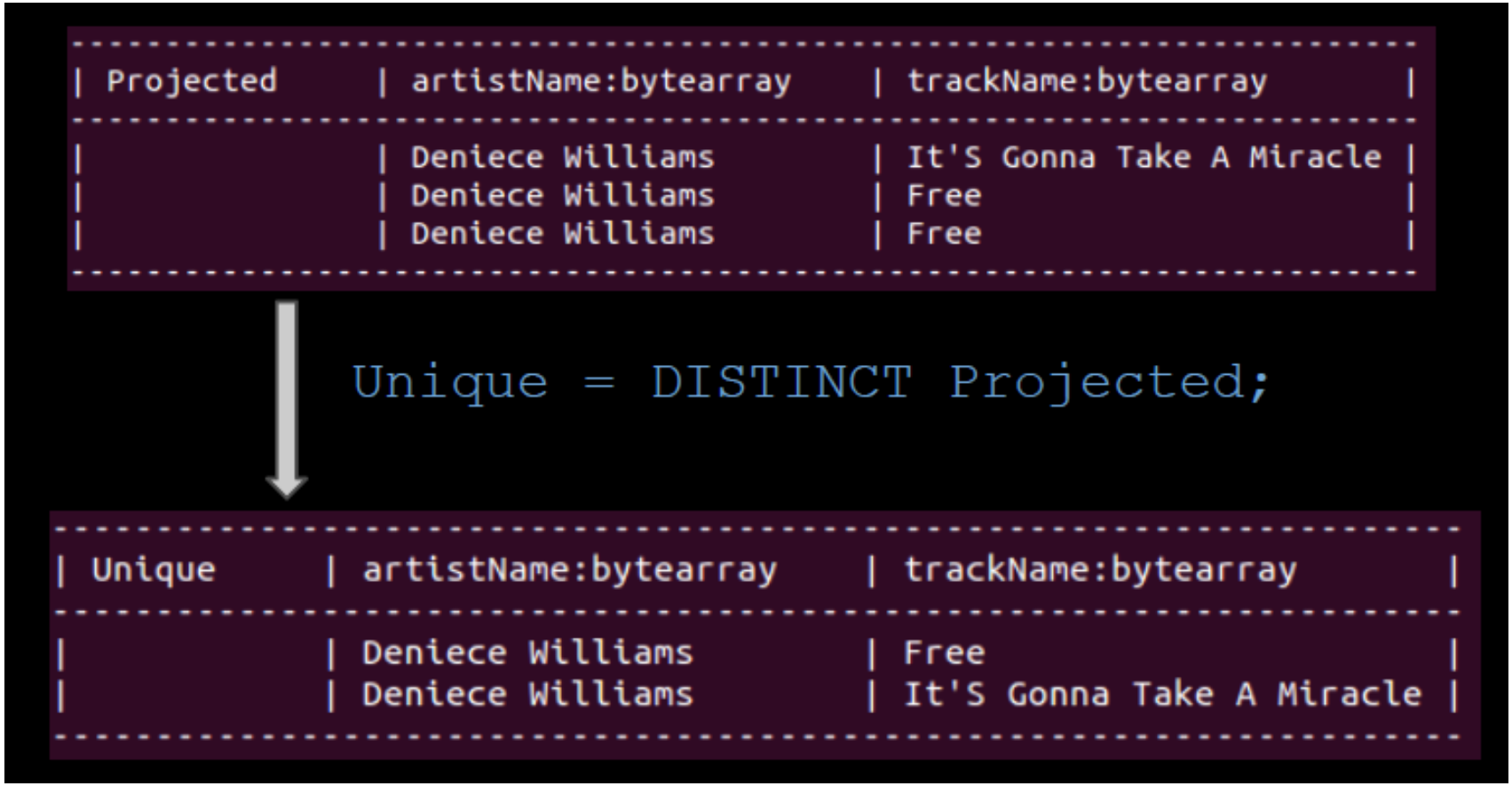
Pig Operations - Projection

- FOREACH ... GENERATE
  - field referenced by name or by position (starting from 0)
  - all fields (*) or a range of field can be referenced (..)
  - BinCond (the ternary operator) is supported
  - a custom UDF can be invoked

```
Users = LOAD 'userid-profile.tsv'
            AS (userId, gender, age: int, country, signup);

FOREACH Users GENERATE $0, gender..country,
      (age >= 18 ? true : false) as isAdult, YEAR(signup) AS signupYear,
      (country IS NOT NULL AND country == USA'
      ? 'compatriot' : 'foreigner') AS nationalityStatus;
```

Pig Operations - Deduplication

- DISTINCT removes duplicate records

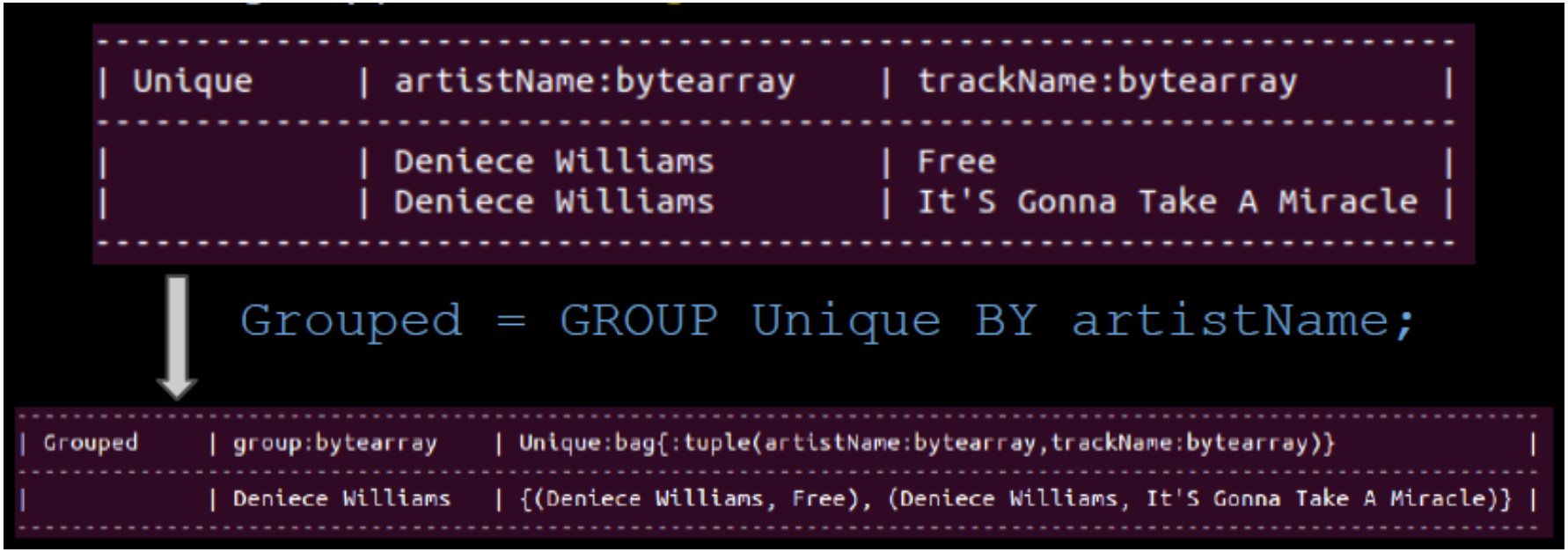


Pig Operations - Deduplication

- DISTINCT removes duplicate records
  - works only on entire records, not on individual fields
  - forces a reduce phase, but optimizes by using the combiner

Pig Operations - Grouping

- GROUPS collects together records with the same key
  - Produces records with two fields: the key (named group) and the bag of collected records



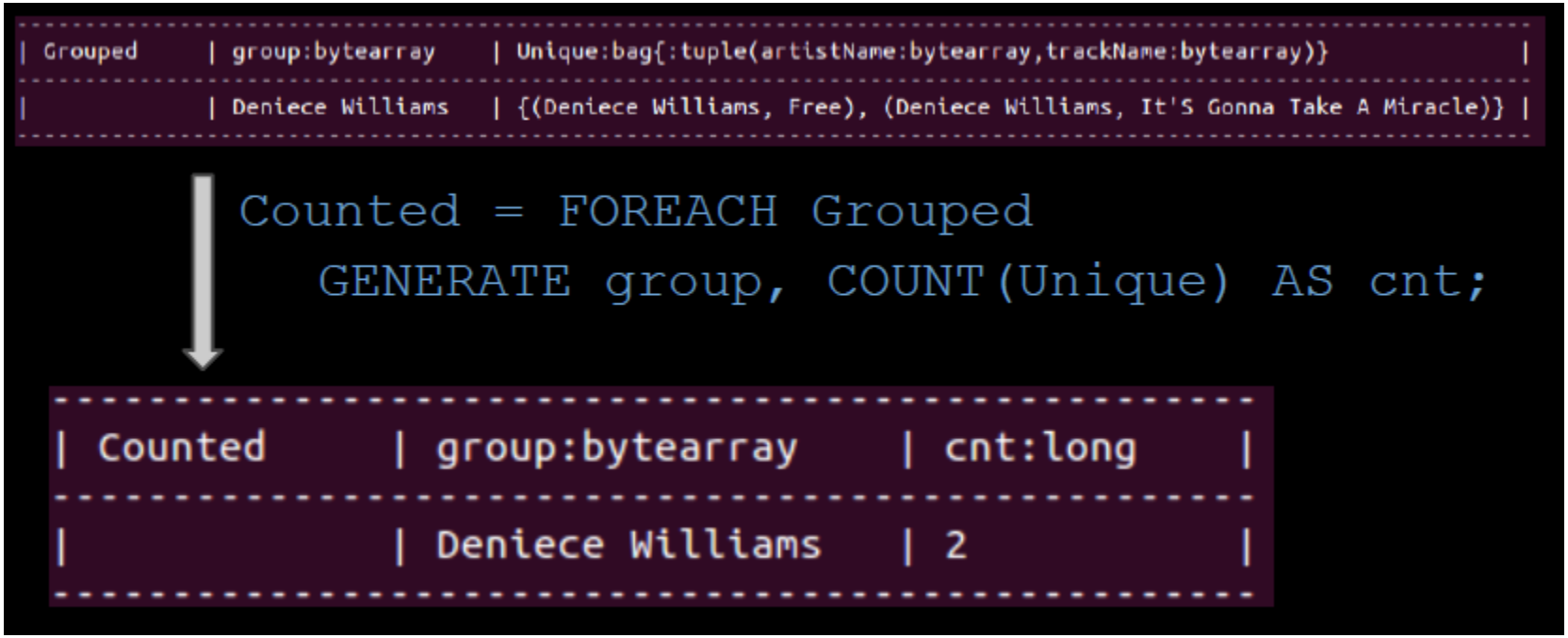
Pig Operations - Grouping

- GROUP collects together records with the same key
  - Support of an expression or user-defined function as the group key
  - Support of grouping on multiple keys
  - Special versions
    - USING 'collected' - avoids a reduce phase
    - GROUP ALL - groups together all of the records into a single group

```
GroupedAll = GROUP Users ALL;
CountedAll = FOREACH GroupedAll GENERATE COUNT
(Users);
```

Pig Operations - Aggregation

- build-in functions e.g. AVG, COUNT, COUNT_STAR, MAX, MIN, SUM
- possibility to implement custom UDFs



Custom Aggregation UDF

```
// src/org/apache/pig/builtin/MyUDF.java
public Long exec(Tuple input) throws IOException {
 try {
  //The data bag is passed to the UDF as the first element of
the
  // tuple.
  DataBag bag = (DataBag)input.get(0);
  Iterator it = bag.iterator();
  while (it.hasNext()){
   Tuple t = (Tuple)it.next();
   // Don't count nulls or empty tuples
   if (t != null && t.size() > 0 && t.get(0) != null) {
    // your code is here
   }
  }
  return something;
 } catch (Exception e) {
        ...
 }
}
```

Pig Operations – Storing Data

- STORE saves the results
- When storing to the filesystem, it creates a directory with part files rather than a single file
- Usually mirrors LOAD

```
STORE Counted INTO 'songsPerArtists';
```

Pig Operations – Diagnostic

- DESCRIBE returns the schema of a relation

```
Projected = FOREACH Songs GENERATE artistName;
Grouped = GROUP Projected BY artistName;
DESCRIBE Grouped;
Grouped: {group: bytearray,Projected: {(artistName: bytearray)}}
```

- DUMP displays results to screen
  - results are not persisted anywhere
  - should not be used in production scripts

Example 1
Find number of tracks per artists

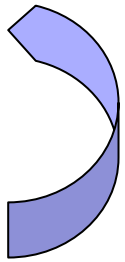
- Runs as two MapReduce jobs
  - Job1: Songs, Projected, Unique
  - Job2: Grouped, Counted

```
--Load the input file
Songs = LOAD 'whug/data/lastfm-dataset-1K/userid-track.tsv'
        AS (userId, timestamp, mbArtistId,
        artistName, mbTrackId, trackName);

-- Remain only artist and track names
Projected = FOREACH Songs GENERATE artistName, trackName;

-- Find unique records
Unique = DISTINCT Projected;

-- Group records by artist name
Grouped = GROUP Unique BY artistName;
-- Count the number of songs per artist
Counted = FOREACH Grouped GENERATE group, COUNT(Unique) AS cnt;

-- Displays a step-by-step execution of a sequence of statements
ILLUSTRATE Counted;
```

Pig Optimizer Example

- Pig optimizer may rearrange statements

```
A = load 'students' as (name, age, gpa);
B = group A by age;
C = foreach B generate group, COUNT (A);
D = filter C by group < 30;
```



- But programmer can make it more efficient (and force combiner)

```
A = load 'students' as (name, age, gpa);
B = filter A by age < 30;
C = group B by age;
D = foreach C generate group, COUNT (B);
```

Pig Optimizer Example

- can merge together two foreach statements

```
-- Original code:
A = LOAD 'file.txt' AS (a, b, c);
B = FOREACH A GENERATE a+b AS u, c-b AS v;
C = FOREACH B GENERATE $0+5, v;

-- Optimized code:
A = LOAD 'file.txt' AS (a, b, c);
C = FOREACH A GENERATE a+b+5, c-b;
```

Pig Optimizer Example

- can simplify the expression in filter statement

```
-- Original code:
B = FILTER A BY ((a0 > 5)
                 OR (a0 > 6 AND a1 > 15);

-- Optimized code:
B = FILTER A BY a0 > 5;
```

Pig Multi-Query Execution

- parses the entire script to determine if intermediate tasks can be combined
- Example: one MULTI_QUERY ,MAP_ONLY job

```
Users = LOAD 'users.txt' AS (user, age: int);

Fltrd = FILTER Users BY age >= 15 and age <= 25;
STORE Fltrd INTO 'Fltrd';

Ages = FOREACH Fltrd GENERATE (age >=18 ? true : false) AS isAdult;
SPLIT Ages INTO Adults IF (isAdult == true), Childs IF (isAdult == false);

STORE Ages INTO 'Ages';
STORE Adults INTO 'Adults';
STORE Childs INTO 'Childs';
```

Example 2
Find top 3 popular tracks per country

```
SET default_parallel '$PARALLEL'

-- Contains more than 19M records (~2.35GB)
Songs = LOAD 'whug/data/lastfm-dataset-1K/userid-track.tsv'
        AS (userId, timestamp, mbArtistId,
        artistName, mbTrackId, trackName);
-- Contains 992 records (~37KB)
Users = LOAD 'whug/data/lastfm-dataset-1K/userid-profile.tsv'
        AS (userId, gender, age: int, country, signup);

SongsProj = FOREACH Songs GENERATE userId, artistName, trackName;
UsersProj = FOREACH Users GENERATE userId, country;
-- Select users with known country
UsersCountried = FILTER UsersProj BY (country IS NOT NULL);

-- Join relations on userId using replicated join
Joined = JOIN SongsProj BY userId, UsersCountried BY userId USING 'replicated';
```

Example 2
Find top 3 popular tracks per country

```
Grouped = GROUP Joined BY (country, artistName, trackName);
-- Count track popularity in a given country
Counted = FOREACH Grouped GENERATE FLATTEN(group), COUNT(Joined) AS count;

-- Group by country
Countried = GROUP Counted BY country;
-- Find top tracks for a given country
CountriedTop = FOREACH Countried {
        OrderedSongs = ORDER Counted BY count DESC;
        TopSongs = LIMIT OrderedSongs 3;
        GENERATE FLATTEN(TopSongs);
}

-- Order by country ($0) and count ($3)
CountriedOrd = ORDER CountriedTop BY $0, $3 DESC;
STORE CountriedOrd  INTO 'whug/output/contriedTop$ID';
```

Example 2
Find top 3 popular tracks per country

```
Norway  Tori Amos         Leather 310
Norway  Backstreet Boys Larger Than Life         289
Norway  Bel Canto         Dreaming Girl   257
Peru    Oasis    Falling Down     252
Peru    Franz Ferdinand No You Girls     237
Peru    Depeche Mode     Wrong    228
Poland  Serge Gainsbourg          Je T'Aime...Moi Non Plus          1154
Poland  Dir En Grey      Ryoujoku No Ame 880
Poland  Dir En Grey      Ain'T Afraid To Die      855
Portugal          Dapunksportif    Friends (Come And Go)    437
Portugal          Chris Rea        The Road To Hell         315
Portugal          Queens Of The Stone Age River In The Road         306
Romania Counting Crows   Colorblind       337
Romania Scooter Fire     199
Romania Pink Floyd       Wish You Were Here       185
Russian Federation       The Kinks        A Well Respected Man     244
Russian Federation       Rancid   Roots Radicals   226
Russian Federation       Placebo Protect Me From What I Want      216
Serbia  Zeigeist         Fight With Shattered Mirrors     626
Serbia  Empire Of The Sun         We Are The People        417
Serbia  Madonna Miles Away       205
```

Pig Operations - Joining

- JOIN joins two or more sets

```
SongsProj                                                    UsersCountried

(user_000002,Travis,Sing)                                    (user_000002, Peru)
(user_000002,Travis,The Weight)                              (user_000003, United States)
(user_000002,U2,I Still Haven'T Found What I'M Looking For)
(user_000002,Van Morrison,Someone Like You)


Joined = JOIN SongsProj BY userId, UsersCountried BY
userId USING 'replicated';

Joined: {SongsProj::userId: bytearray,SongsProj::artistName:
bytearray,SongsProj::trackName: bytearray,UsersCountried::userId:
bytearray,UsersCountried::country: bytearray}

(user_000002,Travis,Sing, user_000002, Peru)
(user_000002,Travis,The Weight, user_000002, Peru)
(user_000002,U2,I Still Haven'T Found What I'M Looking For, user_000002, Peru)
(user_000002,Van Morrison,Someone Like You, user_000002, Peru)
```

Pig Operations - Joining

- Good build-in support for joins
- Many different join implementations
- Left, right and full outer joins are supported
- Joining on multiple keys is supported

Pig Operations - Joining

- Merge Join
  - sets are pre-sorted by the join key
- Merge-Sparse Join
  - sets are pre-sorted and one set has few (< 1% of its total) matching keys
- Replicated Join
  - one set is very large, while other sets are small enough to fit into memory
- Skewed Join
  - when a large number of records for some values of the join key is expected

Pig Operations - Flattening

- FLATTEN un-nest a tuple or a bag

```
Grouped = GROUP Joined
      BY (country, artistName, trackName);

((Malta,Radiohead,I Can'T), {
    (user_000440,Radiohead,I Can'T,user_000440,Malta),
    (user_000440,Radiohead,I Can'T,user_000440,Malta)
})

Counted = FOREACH Grouped
      GENERATE FLATTEN(group), COUNT(Joined) AS count;
(Malta,Radiohead,I Can'T, 2)
```

Pig Operations - Flattening

```
(a,(b,c))
GENERATE $0,FLATTEN($1)
(a,b,c)

({(a,b),(c,d)})
GENERATE FLATTEN($0)
(a,b)
(c,d)

(Rihanna,{(Disturbia),(We ride),(Sos)})
GENERATE $0, FLATTEN($1)
(Rihanna, Disturbia)
(Rihanna, We ride)
(Rihanna, Sos)
```

Pig Operations - Ordering

- ORDER sorts relation by one or more fields
  - Sorting by maps, tuples or bags produces errors
  - Null is taken to be smaller than any possible value for a given type

Pig Operations - Limiting Results

- LIMIT returns limited number of results
  - forces a reduce phase (but has some optimizations)

Pig Operations – Other Operators

- SAMPLE - Selects a random data sample

  Sample = SAMPLE Users 0.1

- SPLIT - Partitions a relation into other relations

```
SPLIT a INTO
    b if id > 3,
    c if id< 5,
    d OTHERWISE;
```

Pig Operations – Other Operators

- UNION - Computes the union of multiple relation

  UNION ONSCHEMA logs2010, logs2011

- CROSS - Computes the cross product of N relations
  - Lots of data

Optimization

Basic Optimization Rules

- Filter early and often
  - apply filters as early as possible to reduce the amount of data processed
  - do not apply filter, if the cost of applying filter is very high and only a small amount of data is filtered out
  - remove NULLs before JOIN

Basic Optimization Rules

- Use the right data type
  - Pig assumes the type of double for numeric computations
  - specify the real type to speed upf arithmetic computation (even 2x speedup for queries like bellow)
  - early error detection

```
--Query 1 (NOT optimal)
A = load 'myfile' as (t, u, v);
B = foreach A generate t + u;
--Query 2 (optimal)
A = load 'myfile' as (t: int, u:int, v);
B = foreach A generate t + u;
```

Basic Optimization Rules

- Use the right JOIN implementation

- Based on Example 2
  - Replicated Join - 2m32.819s
  - Merge Join - 2m53.051s
  - Regular Join - 3m8.193s
  - Skewed Join - 4m11.849s

Basic Optimization Rules

- Combine small input files
  - separate map is created for each file
  - may be inefficient if there are many small files
  - pig.maxCombinedSplitSize – specifies the size of data to be processed by a single map. Smaller files are combined until this size is reached
- Based on Example 2
  - pig.maxCombinedSplitSize 64MB - 2m42.977s
  - pig.maxCombinedSplitSize 128MB - 2m38.076s
  - pig.maxCombinedSplitSize 256MB - 3m8.913s

Basic Optimization Rules

- Use LIMIT operator
- Compress the results of intermediate jobs
- Tune MapReduce and Pig parameters

Testing and Diagnostics

- Diagnostics operators
  - DESCRIBE - returns the schema of a relation
  - DUMP - displays results to screen
  - EXPLAIN - displays execution plans
  - ILLUSTRATE - displays a step-by-step execution of a sequence of statements
- PigUnit
  - helps to perform unit testing and regression testing

Summary

Summary

- We covered the features provided by Pig
- We discussed the major operators in Pig
- We explained the optimizations that can be done for Pig queries

END