Apache Pig

BAS Academy

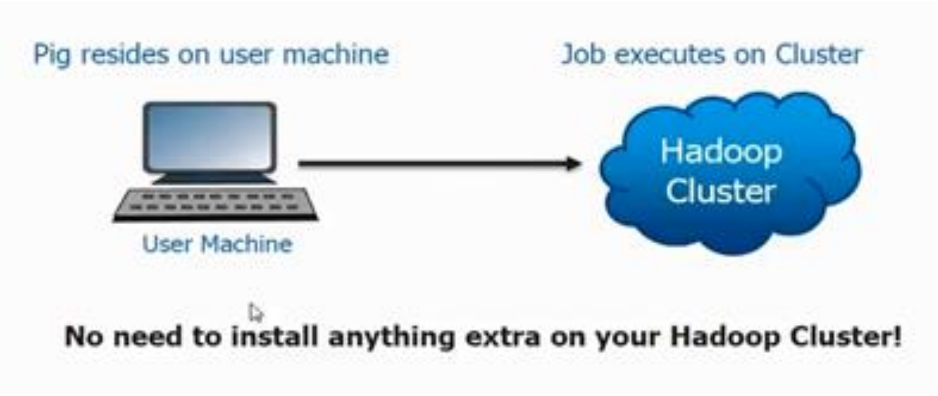
Agenda

- About Pig
- Pig Latin Basics
- Pig Operations
- Advanced Pig Programming
- Hands On

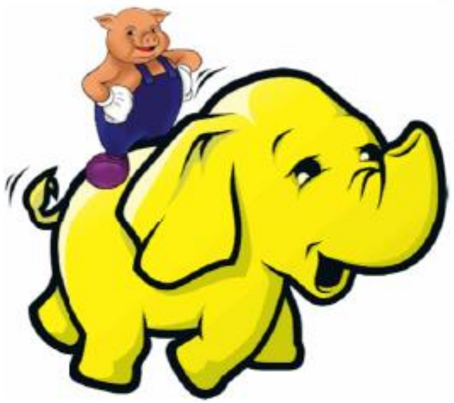
About Pig

What is Pig

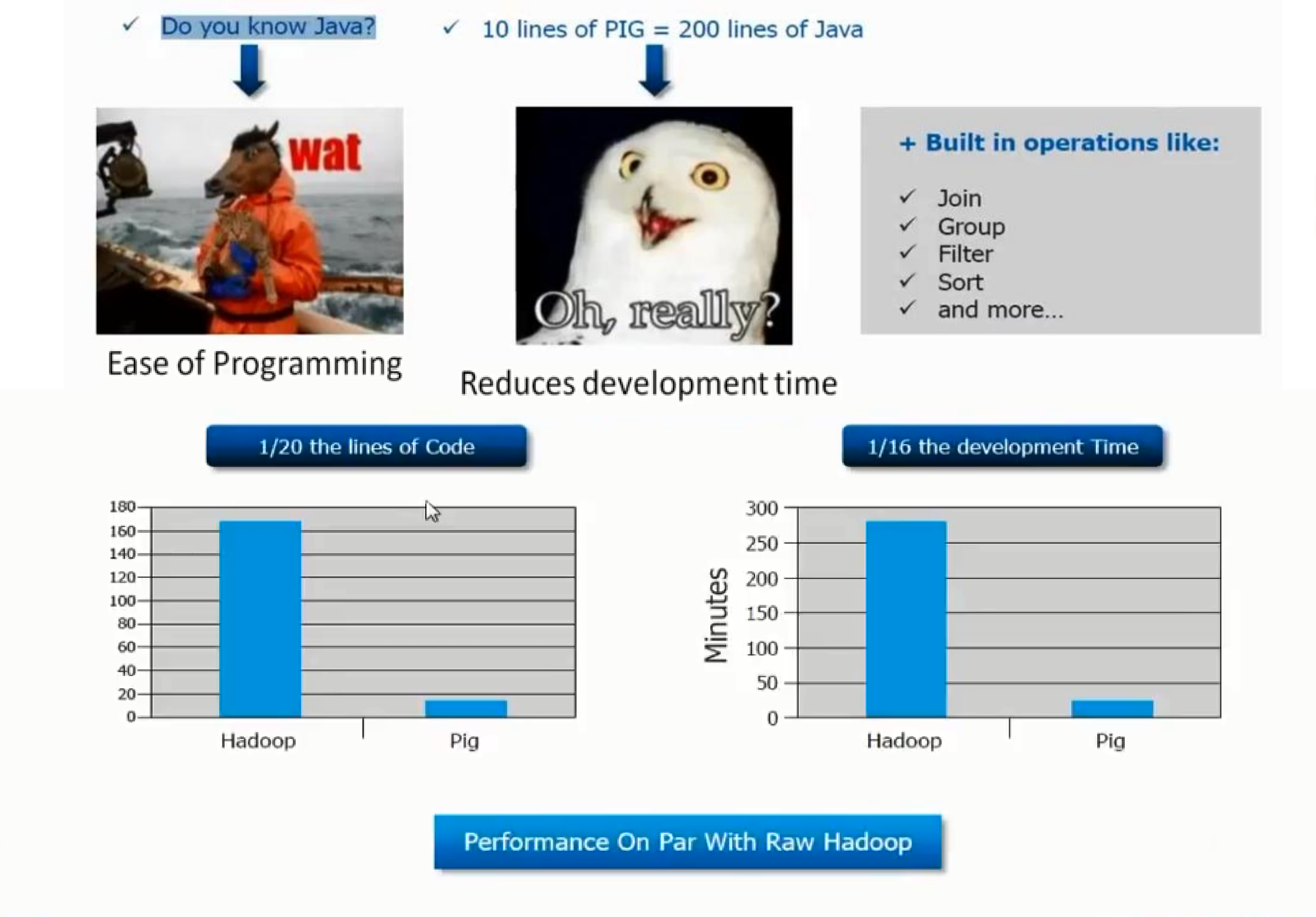
- Framework for analyzing large data sets
- Pig uses high level programming language called Pig Latin
- It runs a MapReduce job with a few lines of code
- Process structured data with schema and unstructured data without schema
- Created by Yahoo



No need to install anything extra on your Hadoop Cluster!



Need for Pig



- ✓ Do you know Java?
- ✓ 10 lines of PIG = 200 lines of Java

Ease of Programming

Reduces development time

+ Built in operations like:

- ✓ Join
- ✓ Group
- ✓ Filter
- ✓ Sort
- ✓ and more…

1/20 the lines of Code

1/16 the development Time

Performance On Par With Raw Hadoop

UDF – User Defined Function Facility

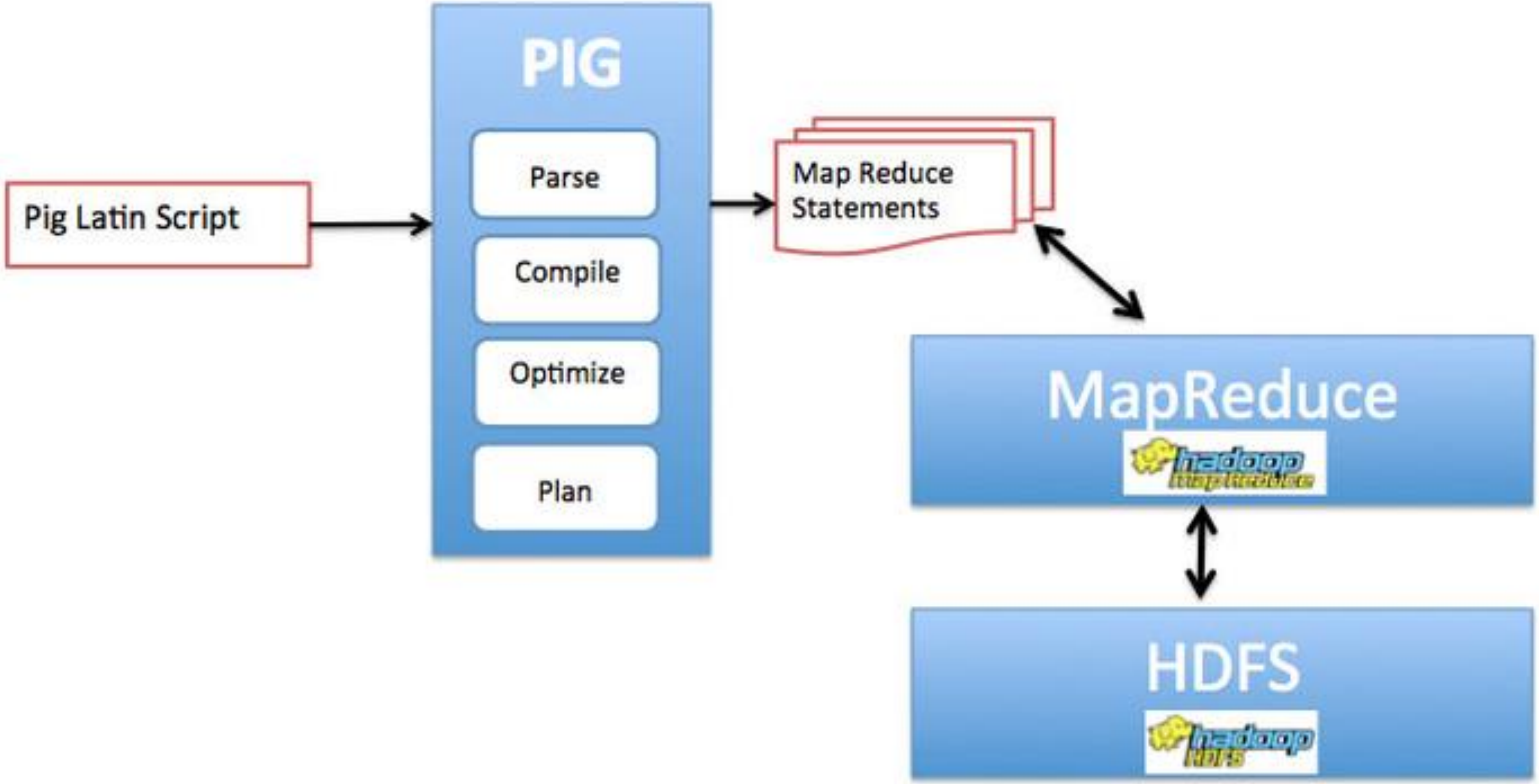
Yahoo and Pig

- Yahoo developed pig to perform search log analytics

- How long is the actual user session?
- How many links does a user click? On or before leaving a website?
- How do click patterns vary in the course of a day/week/month?

- The codes written in Pig Latin automatically get converted to equivalent MapReduce functions

Pig Architecture



Pig Components

- Pig Latin
  - Command based language
  - Designed specifically for data transformation and process

- Execution Environment
  - Pig Latin commands are executed in Local or Hadoop modes

- Pig Compiler
  - Converts Pig Latin to MapReduce
  - Compiler strives to optimize execution

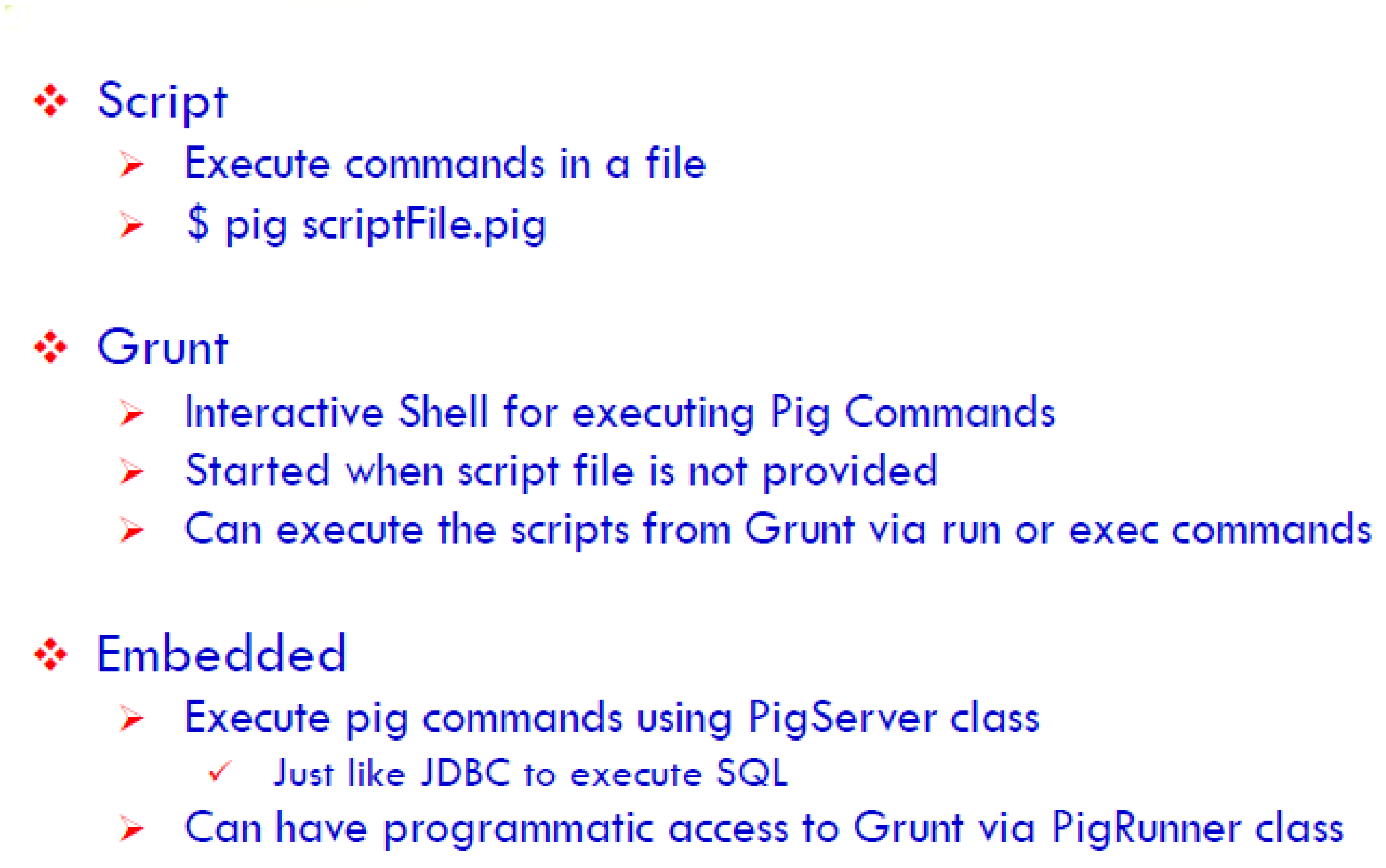
Pig Execution Modes

- Local Mode
  - Executed in single JVM
  - Exclusively works with local file system
  - Great for development, experimentation and prototyping

- Hadoop Mode
  - Also known as MapReduce mode
  - Converts Pig Latin into MapReduce jobs and executes on Hadoop cluster

Executing Pig

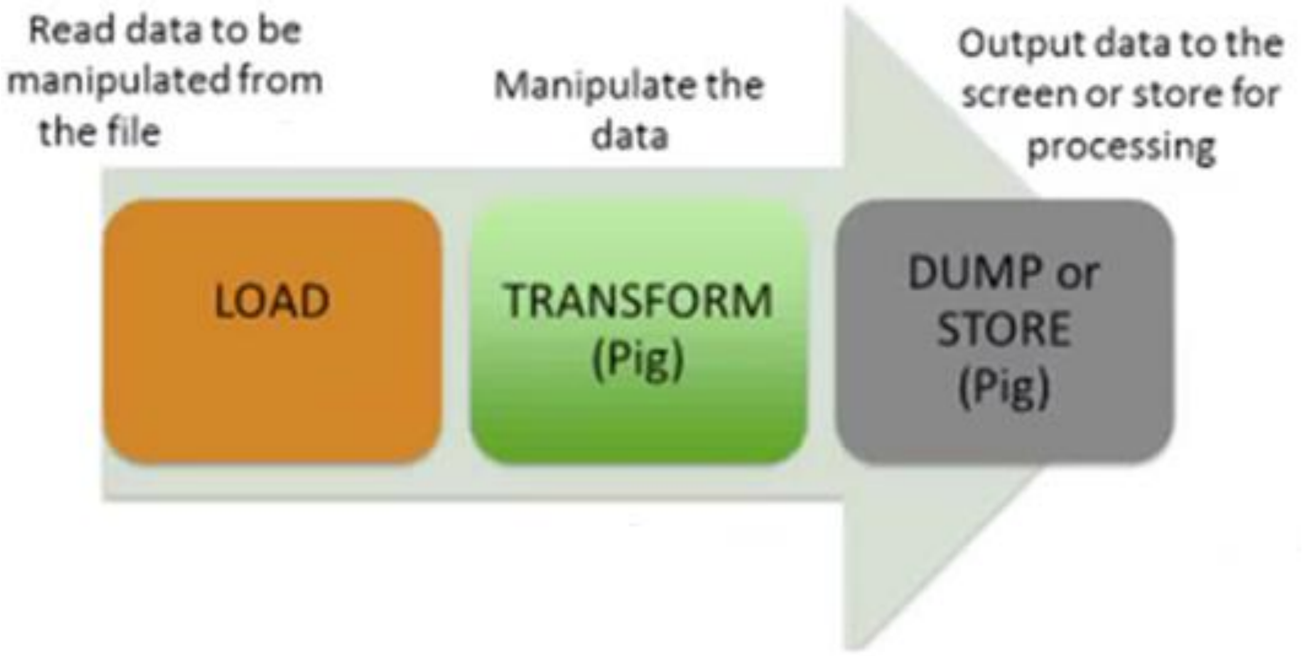
- Pig Latin is a high-level data flow scripting language
- Pig Latin scripts can be executed in one of three ways:



- Script
  - Execute commands in a file
  - $ pig scriptFile.pig
- Grunt
  - Interactive Shell for executing Pig Commands
  - Started when script file is not provided
  - Can execute the scripts from Grunt via run or exec commands
- Embedded
  - Execute pig commands using PigServer class
    - ✓ Just like JDBC to execute SQL
  - Can have programmatic access to Grunt via PigRunner class

Pig Latin Basics

Pig Data Flow



Read data to be manipulated from the file

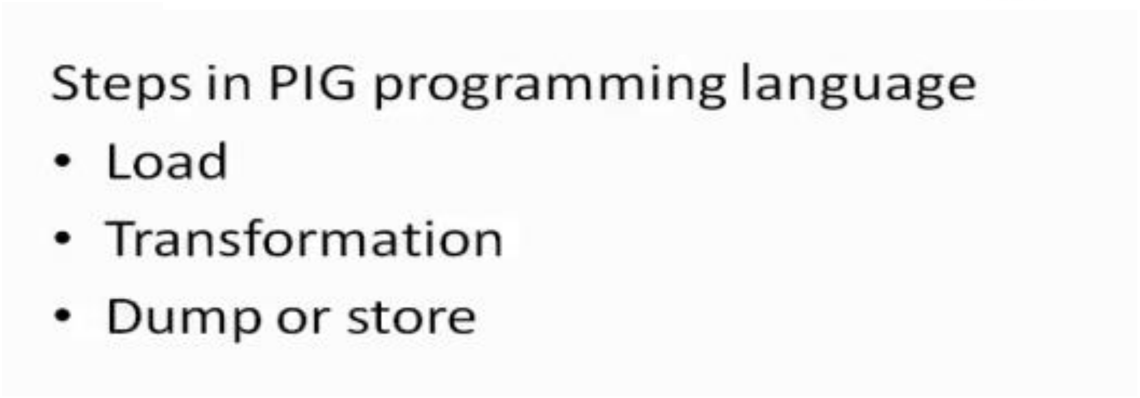
Manipulate the data

Output data to the screen or store for processing

LOAD

TRANSFORM (Pig)

DUMP or STORE (Pig)



Steps in PIG programming language

- Load
- Transformation
- Dump or store

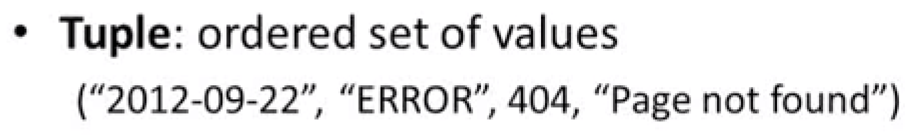
Pig Latin Concepts

Field

- Piece of data **EX: Employee_Name**

Tuple – Similar to row in RDBMS

- Ordered set of fields enclosed in parentheses ()

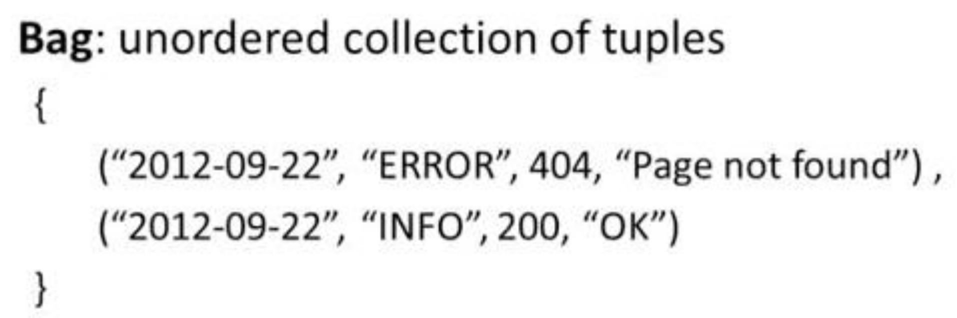


- **Tuple**: ordered set of values

("2012-09-22", "ERROR", 404, "Page not found")

Bag – Similar to table in RDBMS

- Collections of tuples represented in curly braces {}
- Contains tuples with different data types and different no of fields



Bag: unordered collection of tuples

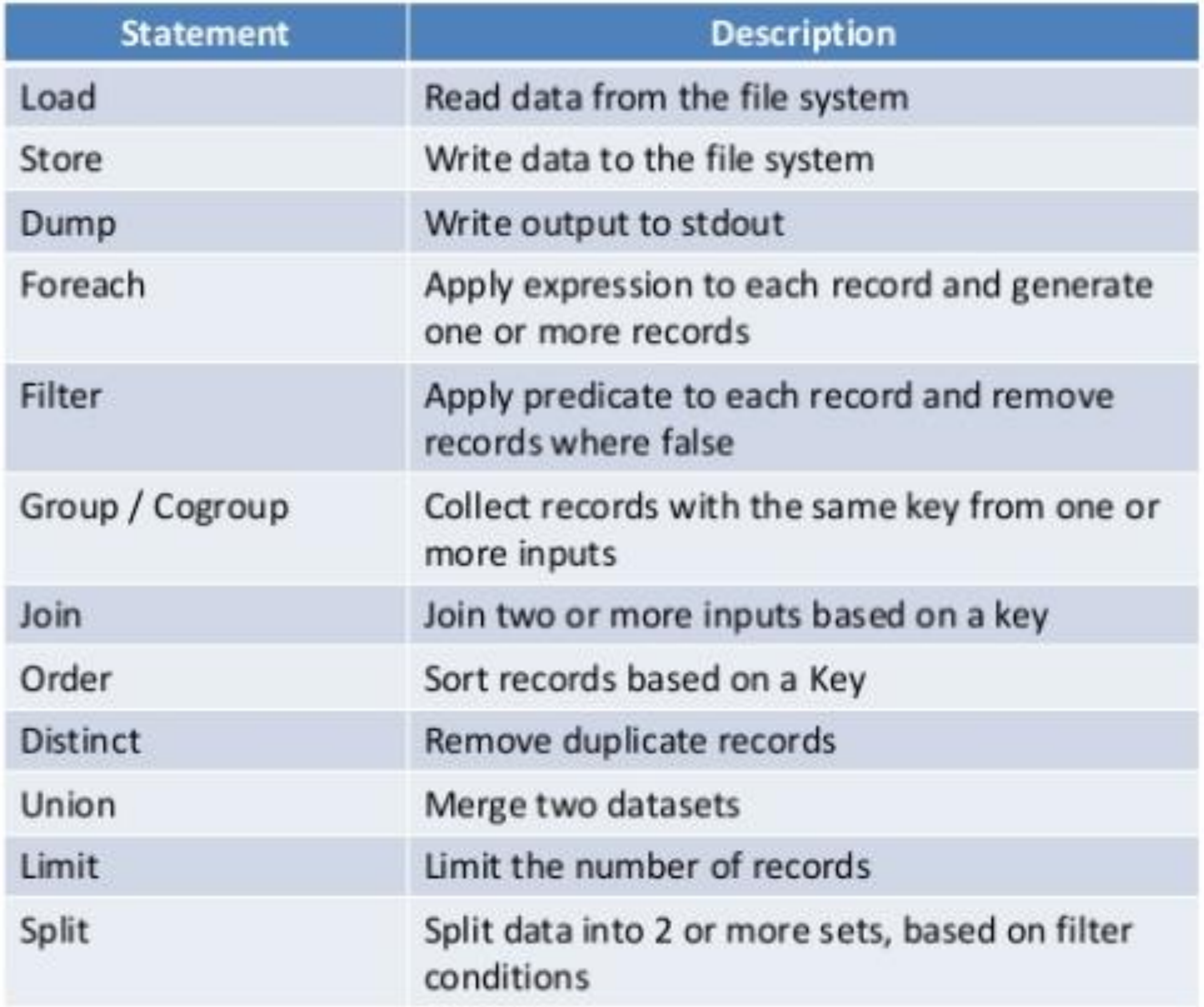
```
{
    ("2012-09-22", "ERROR", 404, "Page not found") ,
    ("2012-09-22", "INFO", 200, "OK")
}
```

Map

- A map is a set of key/value pairs

Ex: [PF#2500,IT#10000,MedCl#6000]

Pig Commands



Statement	Description
Load	Read data from the file system
Store	Write data to the file system
Dump	Write output to stdout
Foreach	Apply expression to each record and generate one or more records
Filter	Apply predicate to each record and remove records where false
Group / Cogroup	Collect records with the same key from one or more inputs
Join	Join two or more inputs based on a key
Order	Sort records based on a Key
Distinct	Remove duplicate records
Union	Merge two datasets
Limit	Limit the number of records
Split	Split data into 2 or more sets, based on filter conditions

- The Pig API has a large collection of built-in functions for performing common tasks and computations.
- /* and an */ are comments
- Double dashes – will comment the line

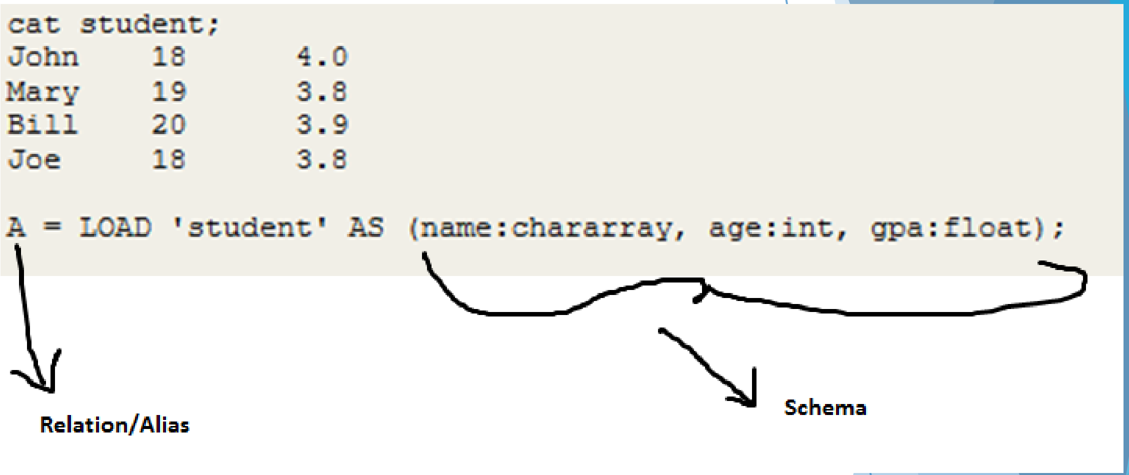
Programming Notions

Relation

- A relation is a bag of tuples with a name.

Alias

- Relations are referred to by name called alias

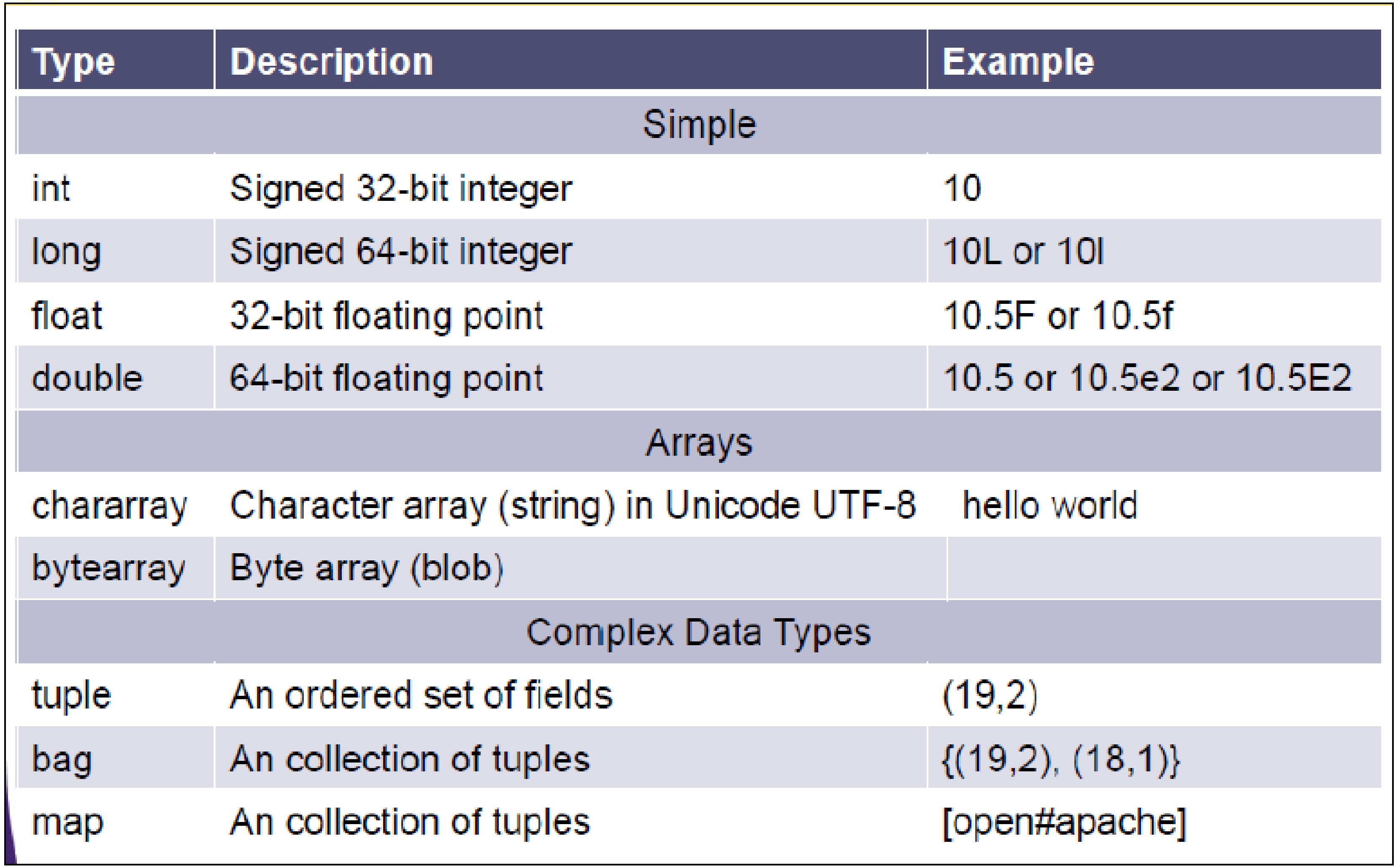


```
cat student;
John     18      4.0
Mary     19      3.8
Bill     20      3.9
Joe      18      3.8

A = LOAD 'student' AS (name:chararray, age:int, gpa:float);
```

Schema

- Schemas enable you to assign names to fields and declare types for fields.
- The schema is typically defined when you load the data using the AS keyword
- Schemas are optional

Pig Schema – Data Types



Type	Description	Example
Simple		
int	Signed 32-bit integer	10
long	Signed 64-bit integer	10L or 10l
float	32-bit floating point	10.5F or 10.5f
double	64-bit floating point	10.5 or 10.5e2 or 10.5E2
Arrays		
chararray	Character array (string) in Unicode UTF-8	hello world
bytearray	Byte array (blob)	
Complex Data Types		
tuple	An ordered set of fields	(19,2)
bag	An collection of tuples	{(19,2), (18,1)}
map	An collection of tuples	[open#apache]

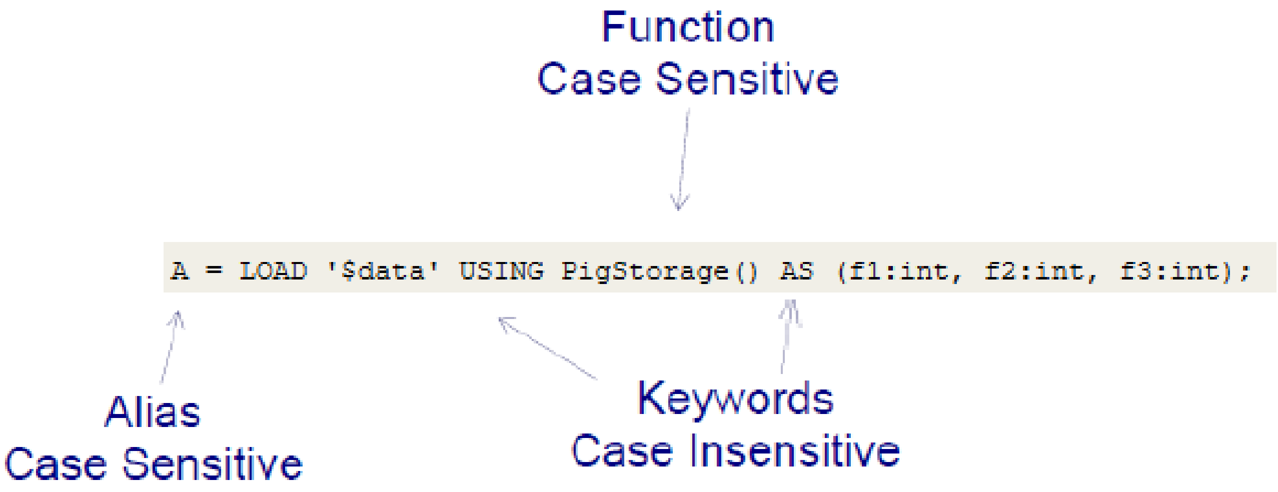
Pig Programming Construct

Case Sensitive

- The names (aliases) of relations and fields are case sensitive
- The names of Pig Latin functions are case sensitive.

Case Insensitive

- The names of and all other Pig Latin keywords are case insensitive.

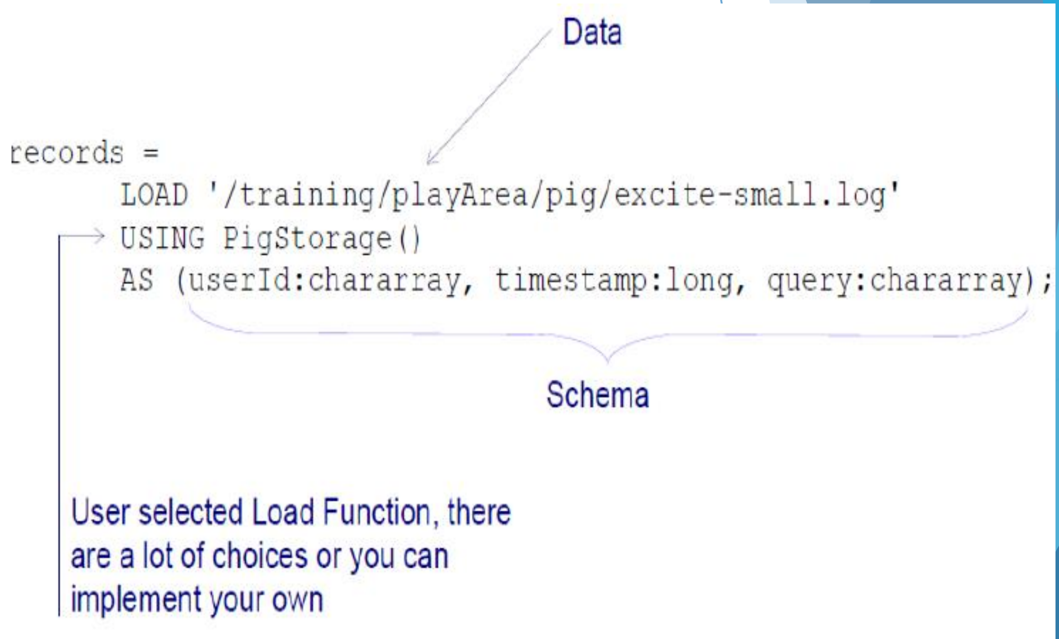


```
A = LOAD '$data' USING PigStorage() AS (f1:int, f2:int, f3:int);
```

Pig Latin – Load Command

LOAD 'data' [USING function] [AS schema];

- data – name of the directory or file
  - Must be in single quotes
- USING – specifies the load function to use
  - By default uses PigStorage which parses each line into fields using a delimiter
    - ✓ Default delimiter is tab ('\t')
    - ✓ The delimiter can be customized using regular expressions
- AS – assign a schema to incoming data
  - Assigns names to fields
  - Declares types to fields

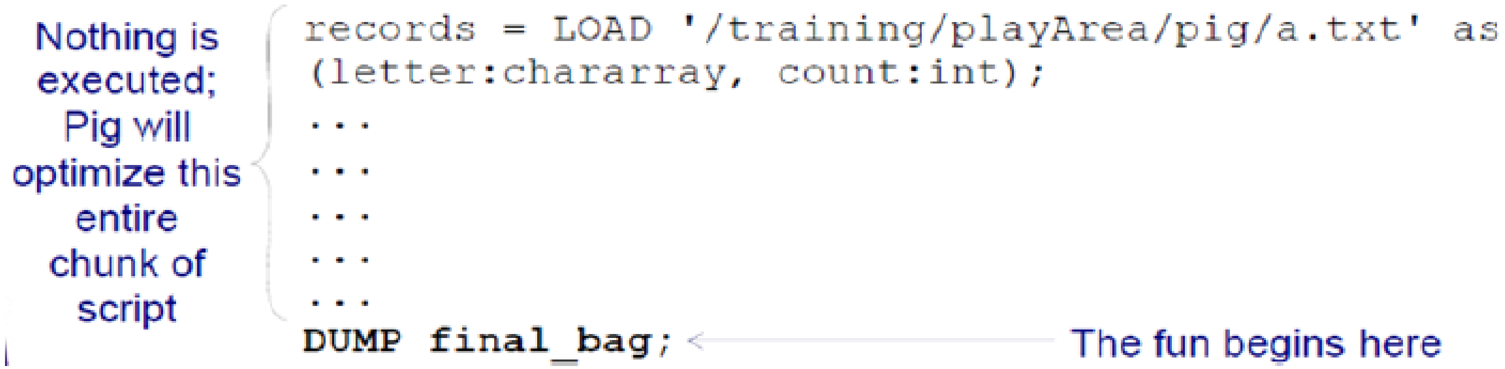


```
records =
       LOAD '/training/playArea/pig/excite-small.log'
       USING PigStorage()
       AS (userId:chararray, timestamp:long, query:chararray);
```

User selected Load Function, there are a lot of choices or you can implement your own

Dump and Store

- No action is taken until DUMP or STORE commands are encountered
  - Pig will parse, validate and analyze statements but not execute them
- DUMP – displays the results to the screen
- STORE – saves results (typically to a file)

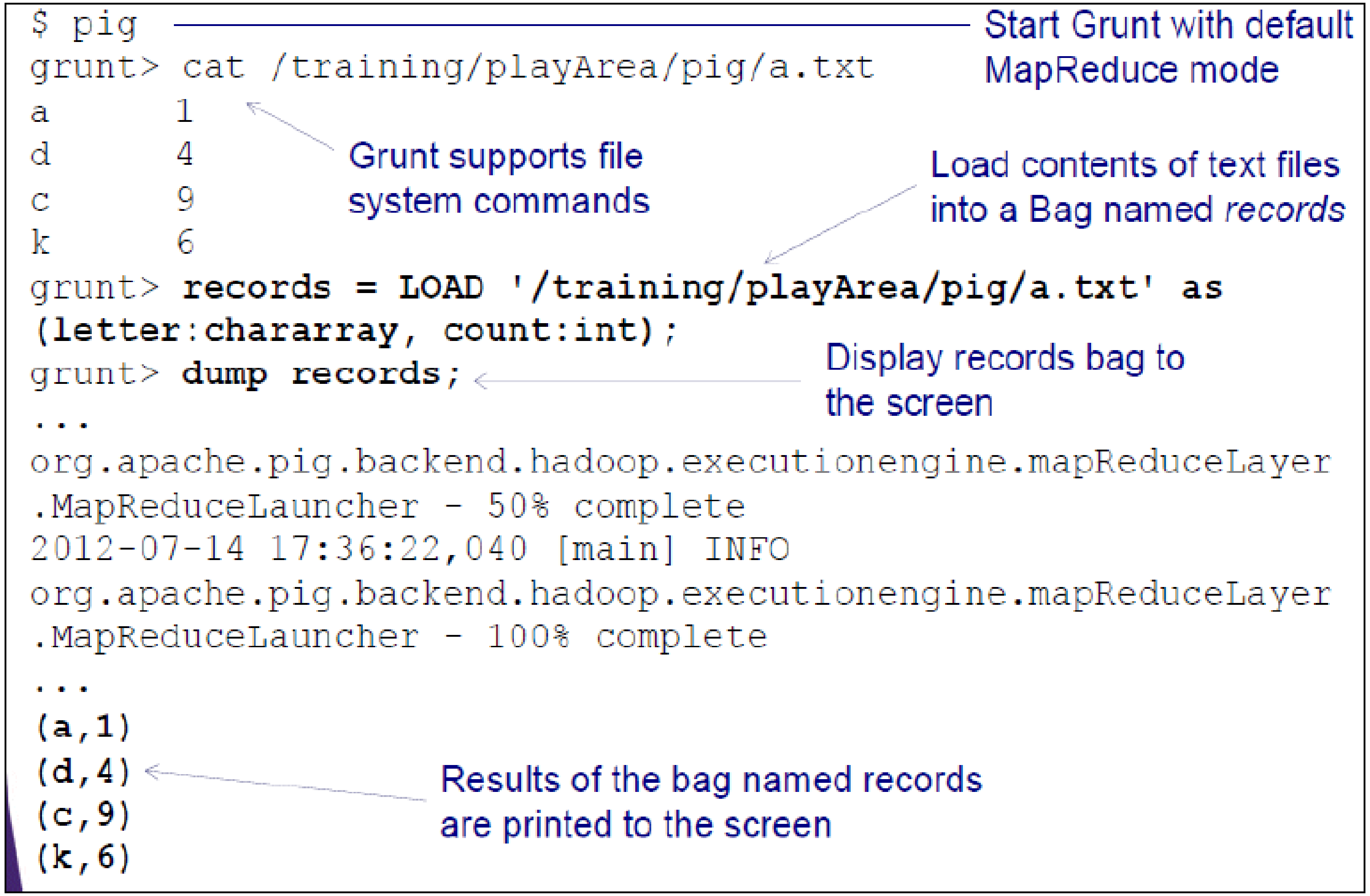


Nothing is executed; Pig will optimize this entire chunk of script

```
records = LOAD '/training/playArea/pig/a.txt' as
(letter:chararray, count:int);
...
...
...
...
...
DUMP final_bag;
```

The fun begins here

Pig Load and Dump - Example



```
$ pig
grunt> cat /training/playArea/pig/a.txt
a       1
d       4
c       9
k       6
grunt> records = LOAD '/training/playArea/pig/a.txt' as
(letter:chararray, count:int);
grunt> dump records;
...
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 50% complete
2012-07-14 17:36:22,040 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer
.MapReduceLauncher - 100% complete
...
(a,1)
(d,4)
(c,9)
(k,6)
```

Start Grunt with default MapReduce mode

Grunt supports file system commands

Load contents of text files into a Bag named *records*

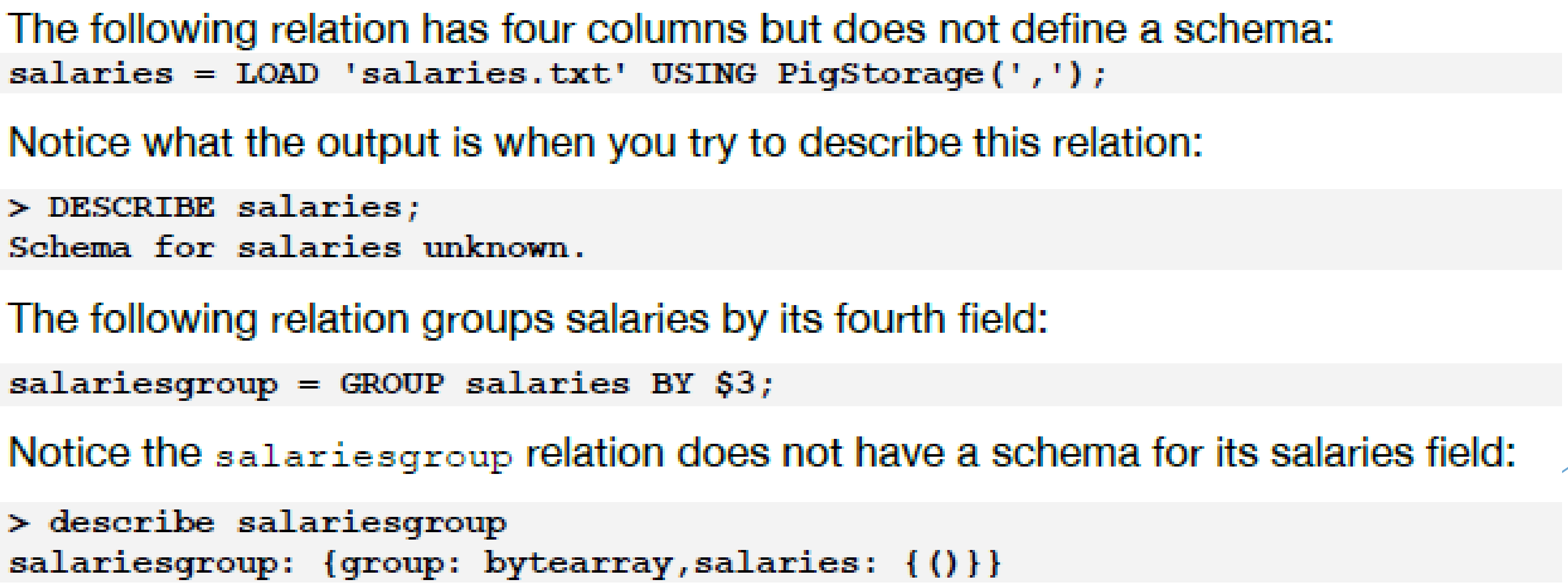
Display records bag to the screen

Results of the bag named records are printed to the screen

Relations without a Schema

- What will happen if the datasets has lot of columns?
- What to do for a data that is not structured?

- If you do not define a schema, Pig will make its best guess as to how the data should be treated
- The fields of a relation are specified using an index that starts at $0.



The following relation has four columns but does not define a schema:

```
salaries = LOAD 'salaries.txt' USING PigStorage(',');
```

Notice what the output is when you try to describe this relation:

```
> DESCRIBE salaries;
Schema for salaries unknown.
```

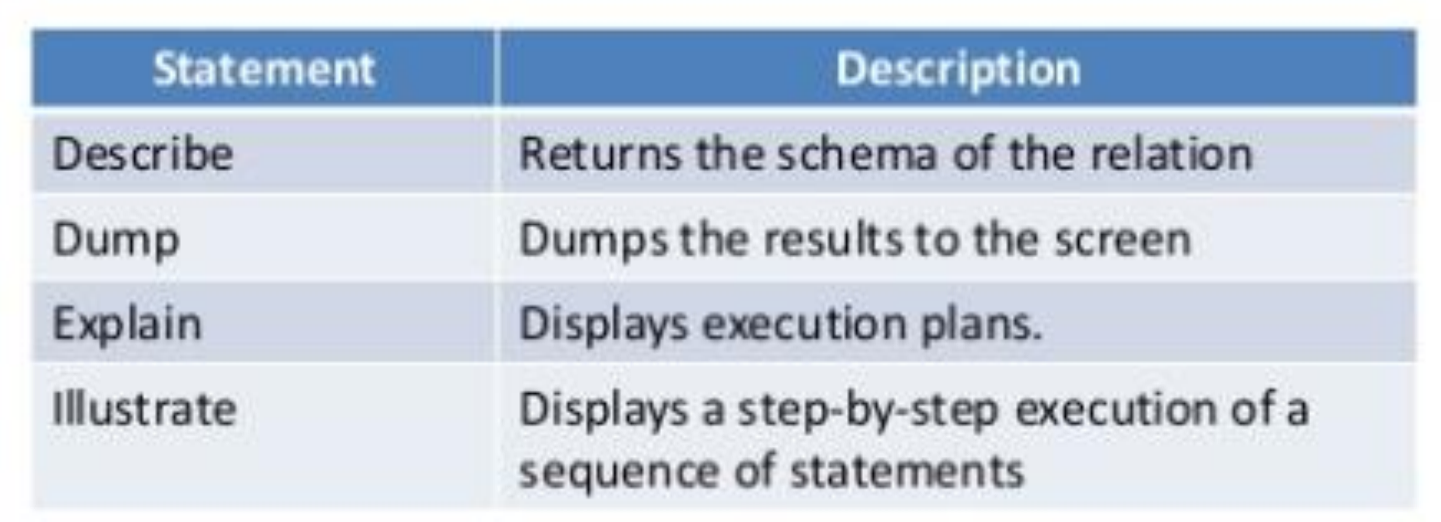
The following relation groups salaries by its fourth field:

```
salariesgroup = GROUP salaries BY $3;
```

Notice the `salariesgroup` relation does not have a schema for its salaries field:

```
> describe salariesgroup
salariesgroup: {group: bytearray,salaries: {()}}
```

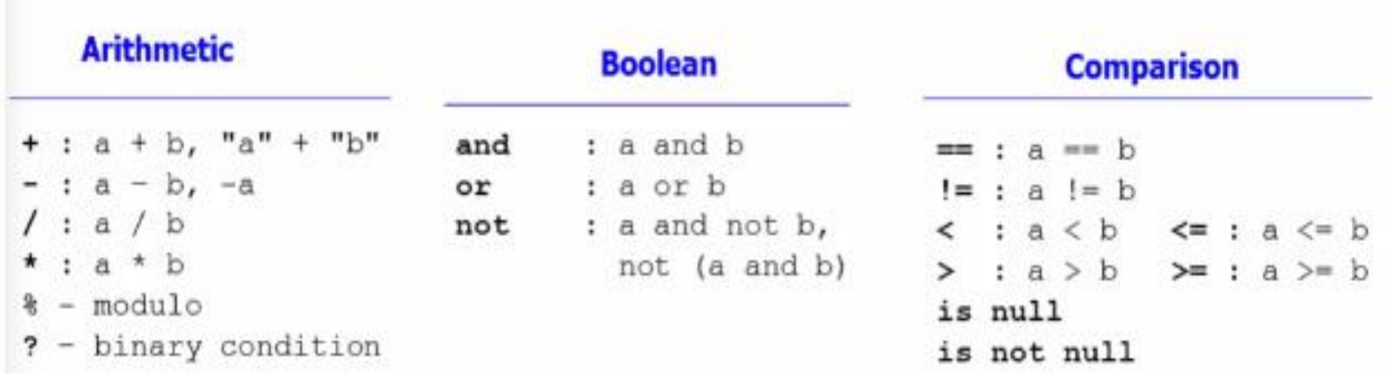
Pig Diagnostic Operators



Statement	Description
Describe	Returns the schema of the relation
Dump	Dumps the results to the screen
Explain	Displays execution plans.
Illustrate	Displays a step-by-step execution of a sequence of statements

- Pig comes with a set of built in functions (Eval, Math, String, Date/Time, Tuple/Bag functions, Load/Store functions).
- Refer https://pig.apache.org/docs/r0.11.1/func.html

Basic Operators



Arithmetic

```
+ : a + b, "a" + "b"
- : a - b, -a
/ : a / b
* : a * b
% - modulo
? - binary condition
```

Boolean

```
and     : a and b
or      : a or b
not     : a and not b,
          not (a and b)
```

Comparison

```
== : a == b
!= : a != b
<  : a < b    <= : a <= b
>  : a > b    >= : a >= b
is null
is not null
```

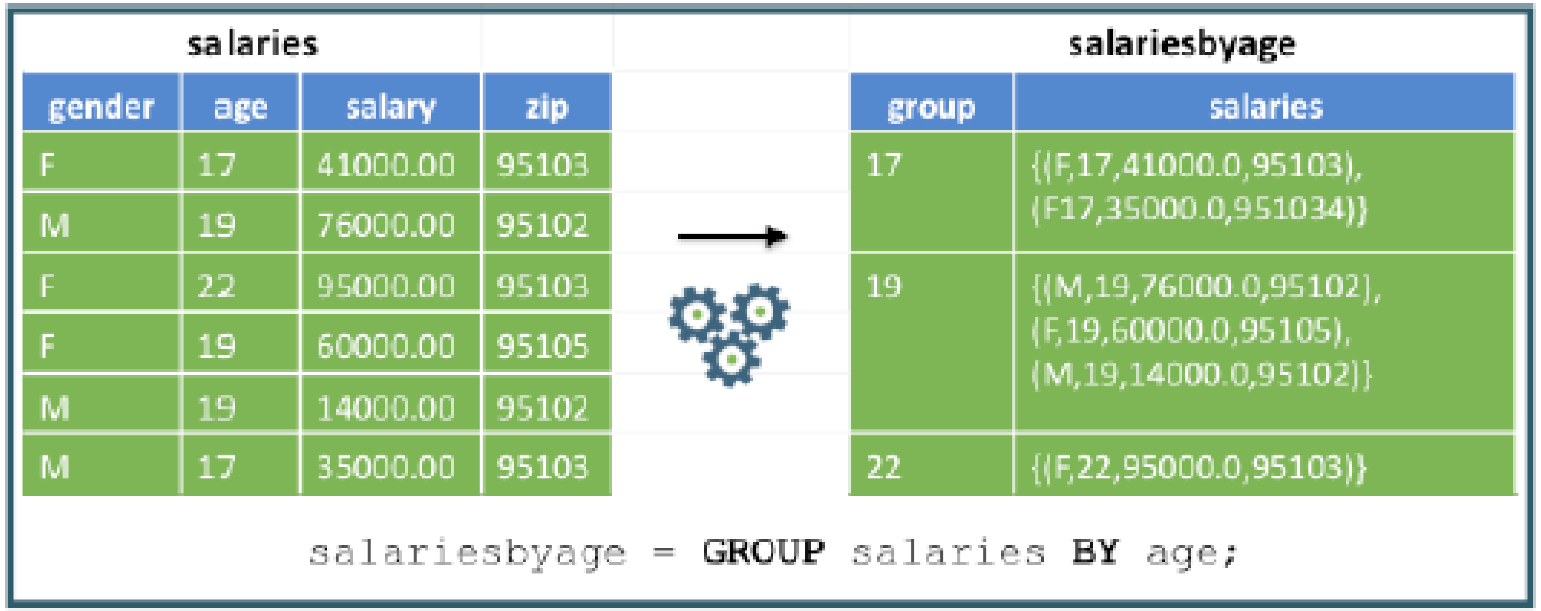
Pig Operations

The Group Operator

- The result of a GROUP operation is a relation that includes one tuple per group.

This tuple contains two fields:

- The first field is named "group" and is the same type as the group key
- The second field takes the name of the original relation and is type bag



salaries

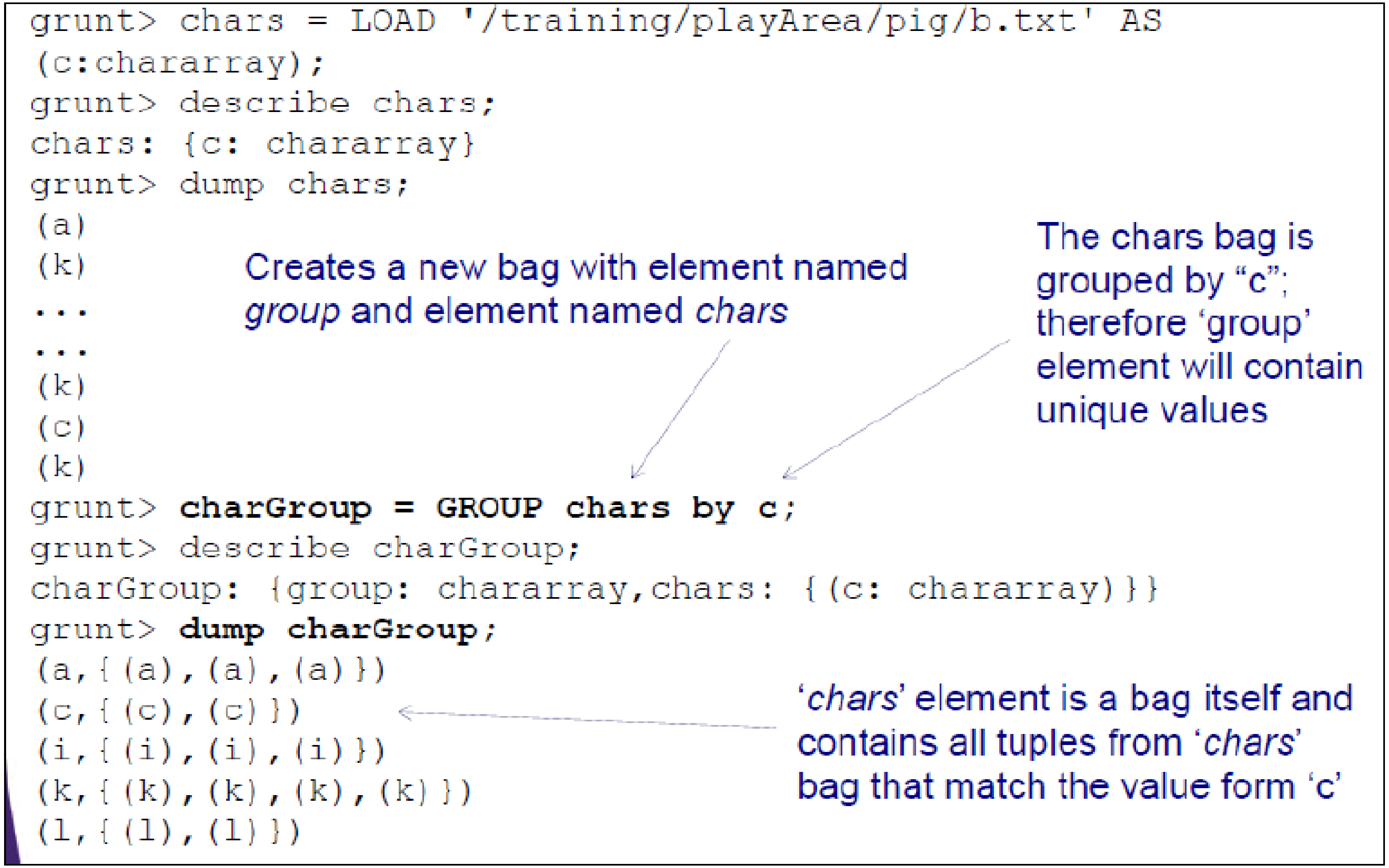
gender	age	salary	zip
F	17	41000.00	95103
M	19	76000.00	95102
F	22	95000.00	95103
F	19	60000.00	95105
M	19	14000.00	95102
M	17	35000.00	95103

salariesbyage

group	salaries
17	{(F,17,41000.0,95103), (F17,35000.0,951034)}
19	{(M,19,76000.0,95102), (F,19,60000.0,95105), (M,19,14000.0,95102)}
22	{(F,22,95000.0,95103)}

```
salariesbyage = GROUP salaries BY age;
```

Group Operator - Example



```
grunt> chars = LOAD '/training/playArea/pig/b.txt' AS
(c:chararray);
grunt> describe chars;
chars: {c: chararray}
grunt> dump chars;
(a)
(k)
...
...
(k)
(c)
(k)
grunt> charGroup = GROUP chars by c;
grunt> describe charGroup;
charGroup: {group: chararray,chars: {(c: chararray)}}
grunt> dump charGroup;
(a,{(a),(a),(a)})
(c,{(c),(c)})
(i,{(i),(i),(i)})
(k,{(k),(k),(k),(k)})
(l,{(l),(l)})
```

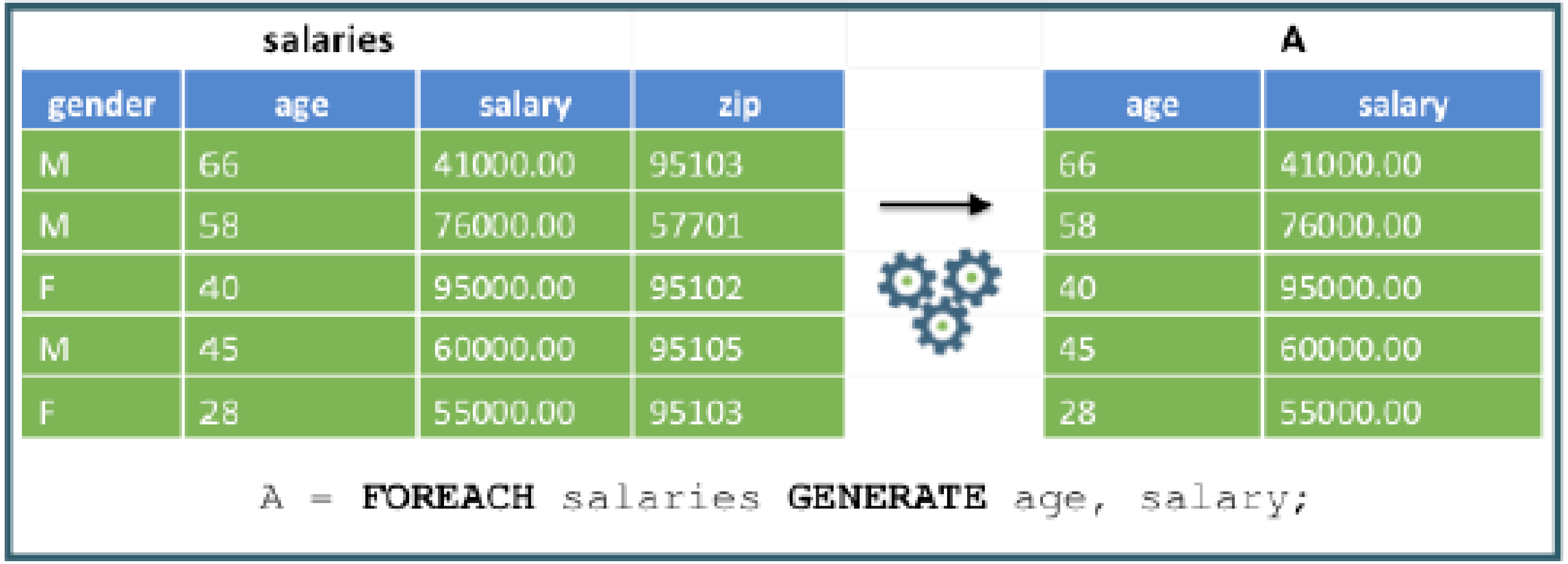
Creates a new bag with element named *group* and element named *chars*

The chars bag is grouped by "c"; therefore 'group' element will contain unique values

'*chars*' element is a bag itself and contains all tuples from '*chars*' bag that match the value form 'c'

The FOREACH GENERATE Operator

- The FOREACH...GENERATE operator transforms records based on a set of expressions that you define.
- It iterates over each element in the bag and produce the result.
- The result of a FOREACH is a new tuple, typically with a different schema.
- FOREACH comes together with functions like COUNT, FLATTEN, CONCAT, etc...



salaries

gender	age	salary	zip
M	66	41000.00	95103
M	58	76000.00	57701
F	40	95000.00	95102
M	45	60000.00	95105
F	28	55000.00	95103

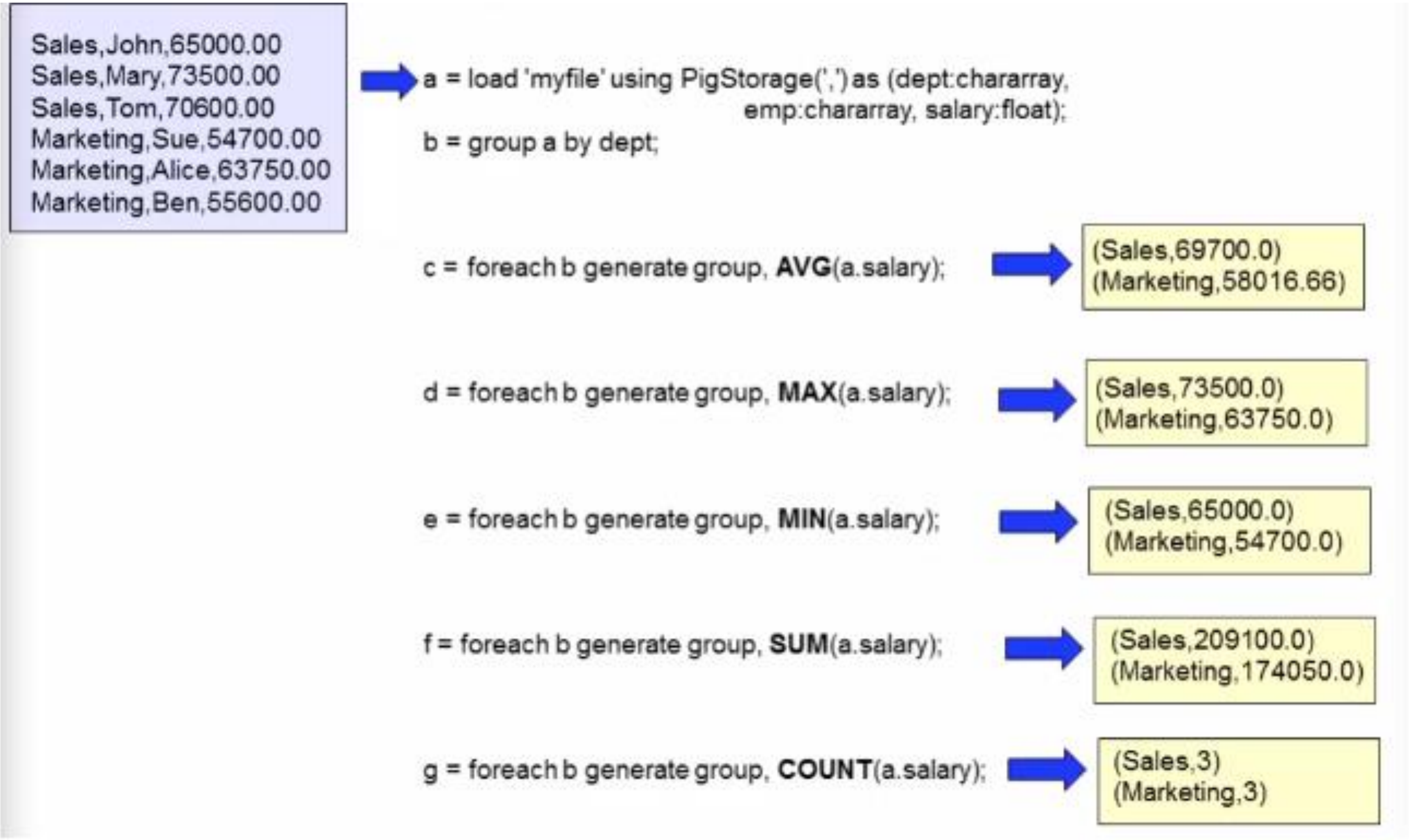
A

age	salary
66	41000.00
58	76000.00
40	95000.00
45	60000.00
28	55000.00

```
A = FOREACH salaries GENERATE age, salary;
```

FOREACH with Function

- FOREACH comes together with functions like COUNT, FLATTEN, CONCAT, etc...



```
Sales,John,65000.00
Sales,Mary,73500.00
Sales,Tom,70600.00
Marketing,Sue,54700.00
Marketing,Alice,63750.00
Marketing,Ben,55600.00
```

```
a = load 'myfile' using PigStorage(',') as (dept:chararray,
                                 emp:chararray, salary:float);
b = group a by dept;
```

```
c = foreach b generate group, AVG(a.salary);
```

```
(Sales,69700.0)
(Marketing,58016.66)
```

```
d = foreach b generate group, MAX(a.salary);
```

```
(Sales,73500.0)
(Marketing,63750.0)
```

```
e = foreach b generate group, MIN(a.salary);
```

```
(Sales,65000.0)
(Marketing,54700.0)
```

```
f = foreach b generate group, SUM(a.salary);
```

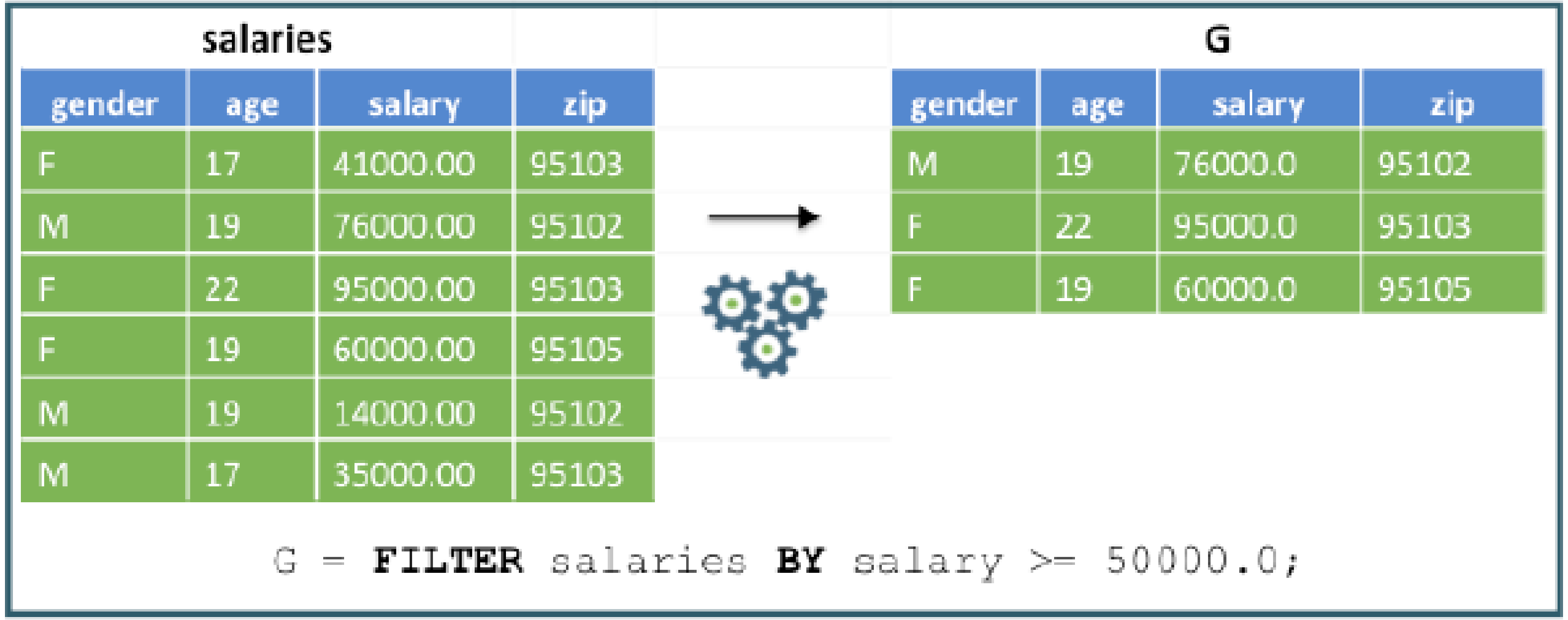
```
(Sales,209100.0)
(Marketing,174050.0)
```

```
g = foreach b generate group, COUNT(a.salary);
```

```
(Sales,3)
(Marketing,3)
```

The Filter Operator

- The FILTER operator selects tuples from a relation based on specified Boolean expressions.
- Conditions can be combined using AND or OR
- The FILTER command does not change the schema of a relation or the structure.



salaries

gender	age	salary	zip
F	17	41000.00	95103
M	19	76000.00	95102
F	22	95000.00	95103
F	19	60000.00	95105
M	19	14000.00	95102
M	17	35000.00	95103

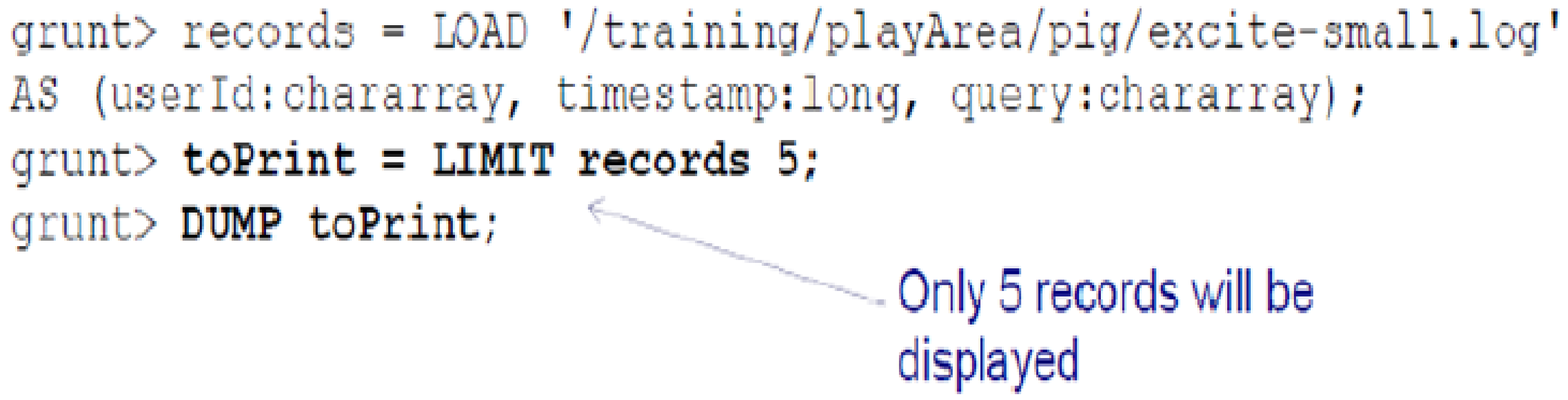
G

gender	age	salary	zip
M	19	76000.0	95102
F	22	95000.0	95103
F	19	60000.0	95105

```
G = FILTER salaries BY salary >= 50000.0;
```

The LIMIT Operator

- The LIMIT command limits the number of output tuples for a relation



```
grunt> records = LOAD '/training/playArea/pig/excite-small.log'
AS (userId:chararray, timestamp:long, query:chararray);
grunt> toPrint = LIMIT records 5;
grunt> DUMP toPrint;
```

Only 5 records will be displayed

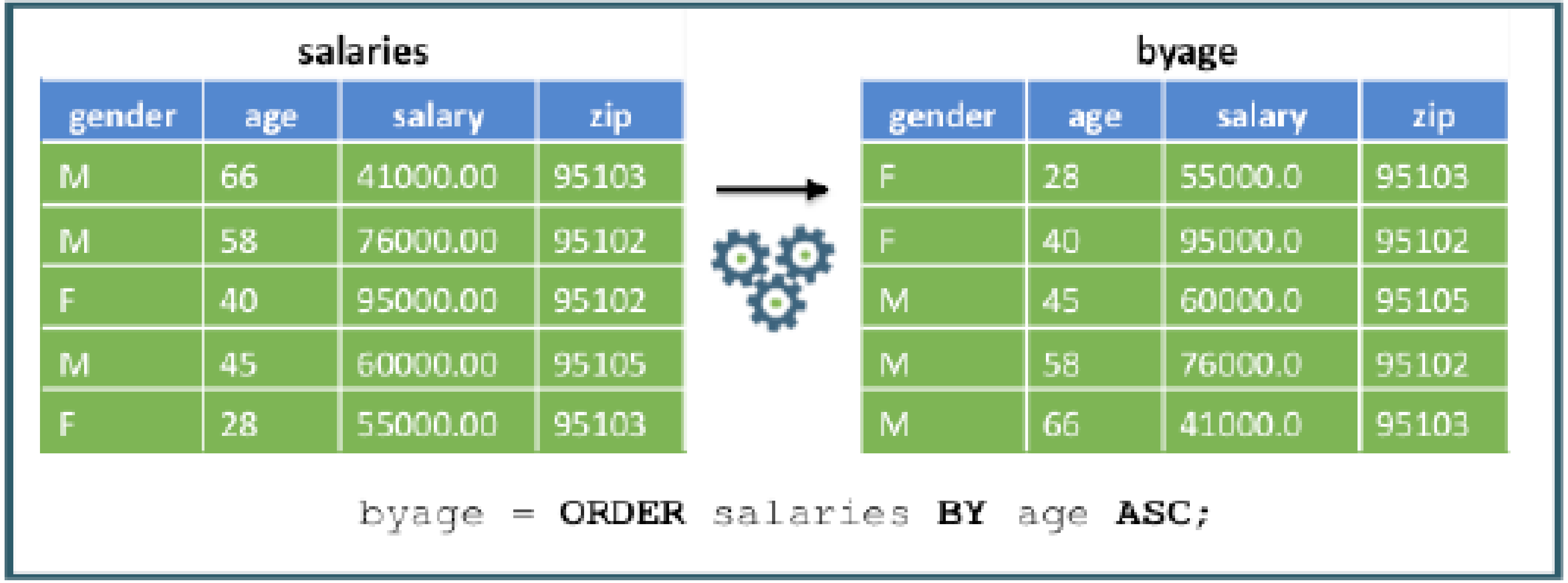
Advanced Pig Programming

Advance Operators

- ORDER BY Operator
- CASE Operator
- DISTINCT Operator
- Use of PARALLEL
- FLATTEN Operator
- COGROUP Operator
- Pig UDF

The ORDER BY Operator

- The ORDER BY command allows you to sort the data in a relation
- You can use DESC or ASC in the BY clause.
- You can also order by multiple fields



salaries

gender	age	salary	zip
M	66	41000.00	95103
M	58	76000.00	95102
F	40	95000.00	95102
M	45	60000.00	95105
F	28	55000.00	95103

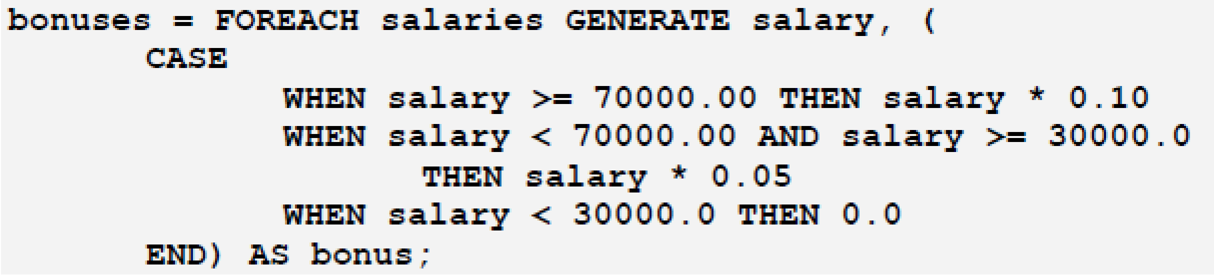
byage

gender	age	salary	zip
F	28	55000.0	95103
F	40	95000.0	95102
M	45	60000.0	95105
M	58	76000.0	95102
M	66	41000.0	95103

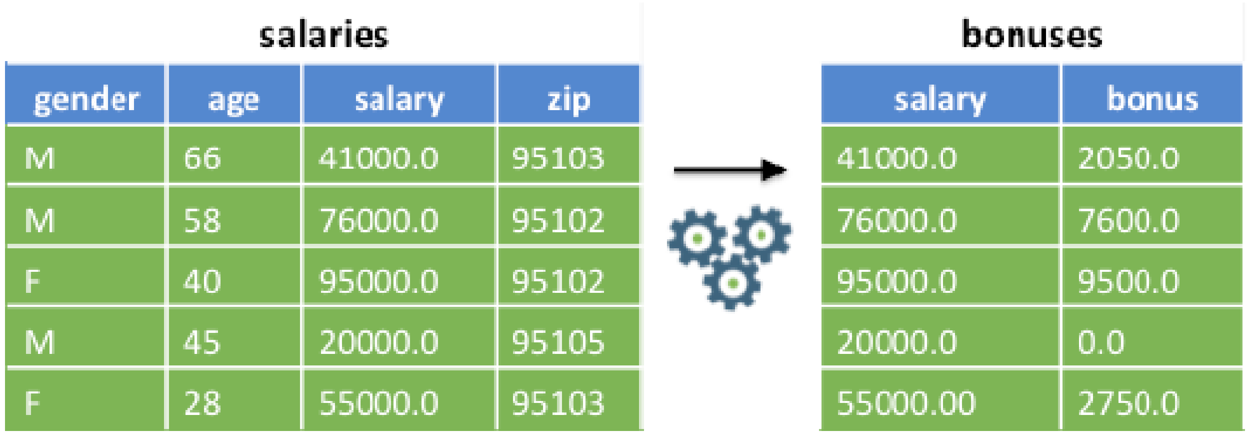
```
byage = ORDER salaries BY age ASC;
```

The CASE Operator

- Pig has a CASE operator that allows you to make decisions within a FOREACH GENERATE statement.



```
bonuses = FOREACH salaries GENERATE salary, (
      CASE
            WHEN salary >= 70000.00 THEN salary * 0.10
            WHEN salary < 70000.00 AND salary >= 30000.0
                  THEN salary * 0.05
            WHEN salary < 30000.0 THEN 0.0
      END) AS bonus;
```



salaries

gender	age	salary	zip
M	66	41000.0	95103
M	58	76000.0	95102
F	40	95000.0	95102
M	45	20000.0	95105
F	28	55000.0	95103

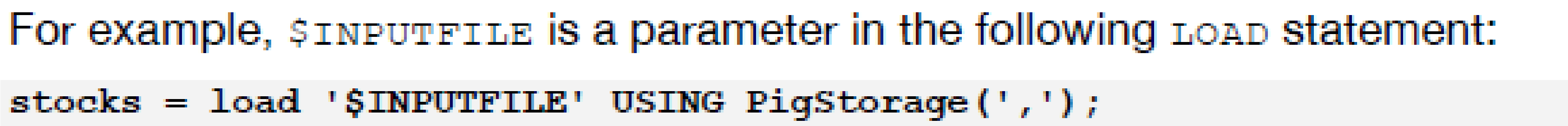
bonuses

salary	bonus
41000.0	2050.0
76000.0	7600.0
95000.0	9500.0
20000.0	0.0
55000.00	2750.0

The CASE Operator

Parameter Substitution

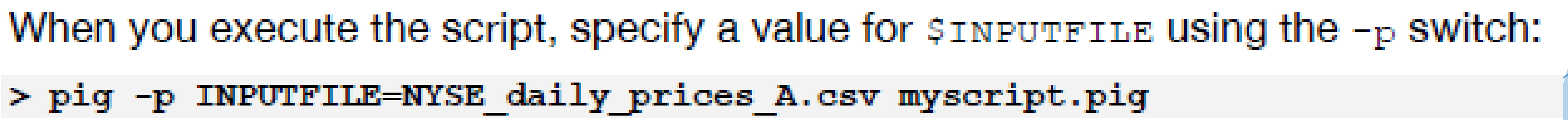
- Pig provides a parameter substitution feature that allows your Pig scripts to refer to values that can be defined at runtime
- A parameter is a value that starts with a dollar sign ($).



For example, $INPUTFILE is a parameter in the following LOAD statement:

```
stocks = load '$INPUTFILE' USING PigStorage(',');
```

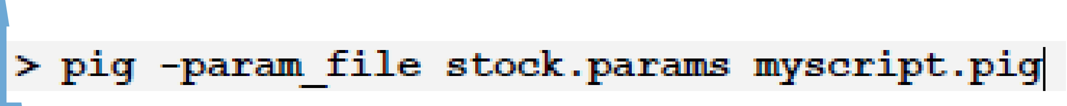
- The above command is stored in the script myscript.pig
- To refer to values at runtime from command line, use -p



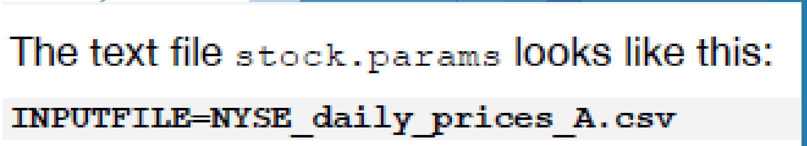
When you execute the script, specify a value for $INPUTFILE using the -p switch:

```
> pig -p INPUTFILE=NYSE_daily_prices_A.csv myscript.pig
```

- To refer to values at runtime from properties file, use -param_file



```
> pig -param_file stock.params myscript.pig
```

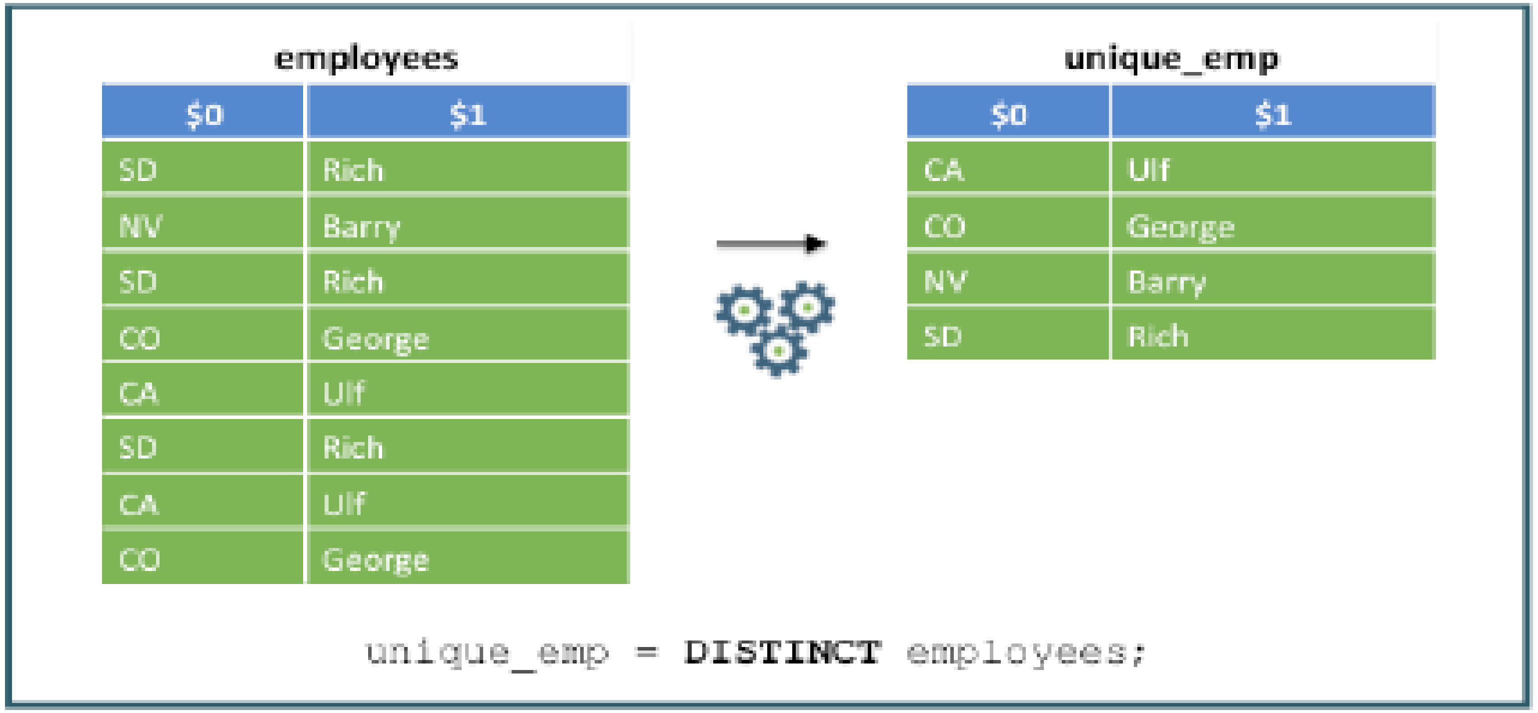


The text file stock.params looks like this:

```
INPUTFILE=NYSE_daily_prices_A.csv
```

The DISTINCT Operator

- The DISTINCT operator removes duplicate tuples in a relation



employees

$0	$1
SD	Rich
NV	Barry
SD	Rich
CO	George
CA	Ulf
SD	Rich
CA	Ulf
CO	George

unique_emp

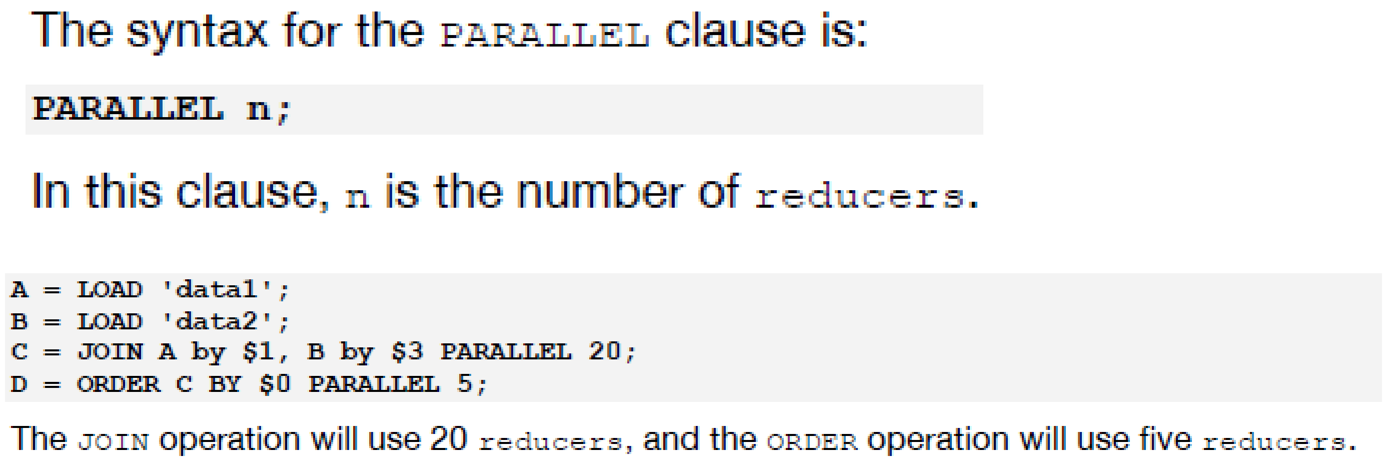
$0	$1
CA	Ulf
CO	George
NV	Barry
SD	Rich

```
unique_emp = DISTINCT employees;
```

The DISTINCT Operator

Using PARALLEL

- The PARALLEL operator is a clause used to determine the number of reducers in the subsequent MapReduce job for that particular operation



The syntax for the `PARALLEL` clause is:

```
PARALLEL n;
```

In this clause, `n` is the number of `reducers`.

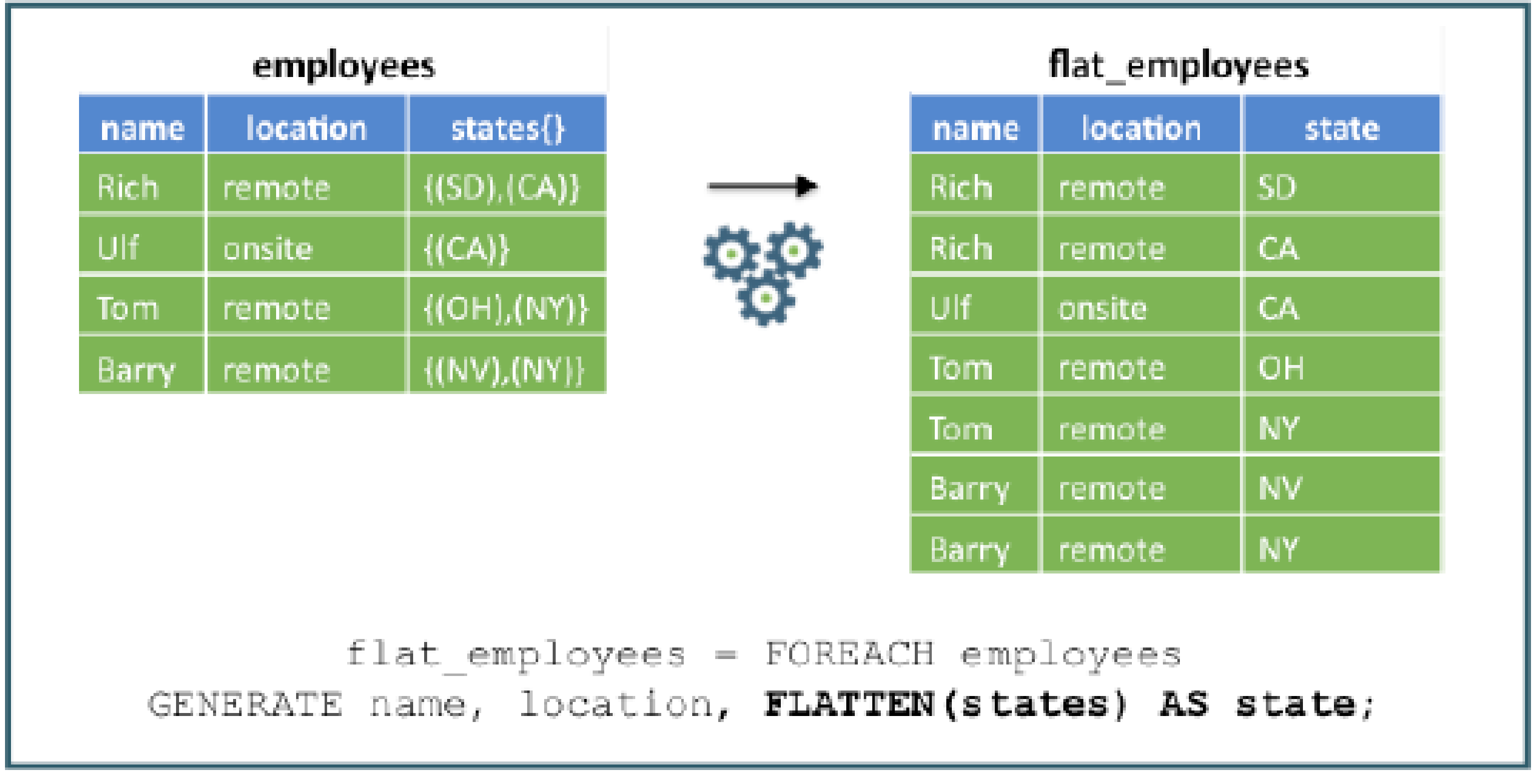
```
A = LOAD 'data1';
B = LOAD 'data2';
C = JOIN A by $1, B by $3 PARALLEL 20;
D = ORDER C BY $0 PARALLEL 5;
```

The `JOIN` operation will use 20 `reducers`, and the `ORDER` operation will use five `reducers`.

- Some Pig operators do not require a reduce phase; these are LOAD, FOREACH, FILTER
- Some operators have a reduce phase, like GROUP, ORDER BY, DISTINCT, JOIN and COGROUP.

The FLATTEN Operator

- The FLATTEN operator removes the nesting of nested tuples and bags.
- To invoke a FLATTEN function, pass the tuple or bag that you want to flatten



employees

name	location	states{}
Rich	remote	{(SD),(CA)}
Ulf	onsite	{(CA)}
Tom	remote	{(OH),(NY)}
Barry	remote	{(NV),(NY)}

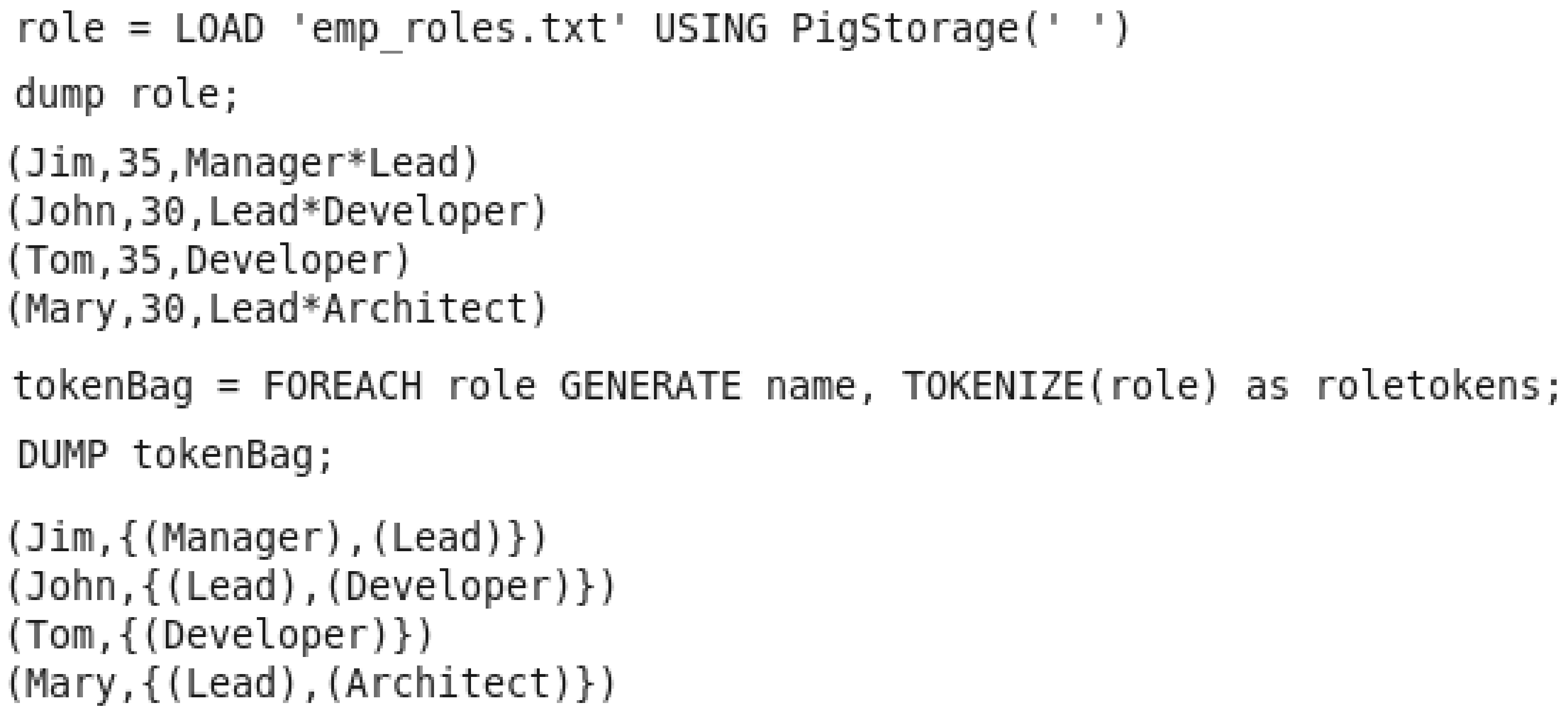
flat_employees

name	location	state
Rich	remote	SD
Rich	remote	CA
Ulf	onsite	CA
Tom	remote	OH
Tom	remote	NY
Barry	remote	NV
Barry	remote	NY

```
     flat_employees = FOREACH employees
GENERATE name, location, FLATTEN(states) AS state;
```

The TOKENIZE Operator

- Splits a string and outputs a bag of words.
- TOKENIZE() function accepts double quote, coma, parenthesis, star as delimiters



```
role = LOAD 'emp_roles.txt' USING PigStorage(' ')
dump role;
```

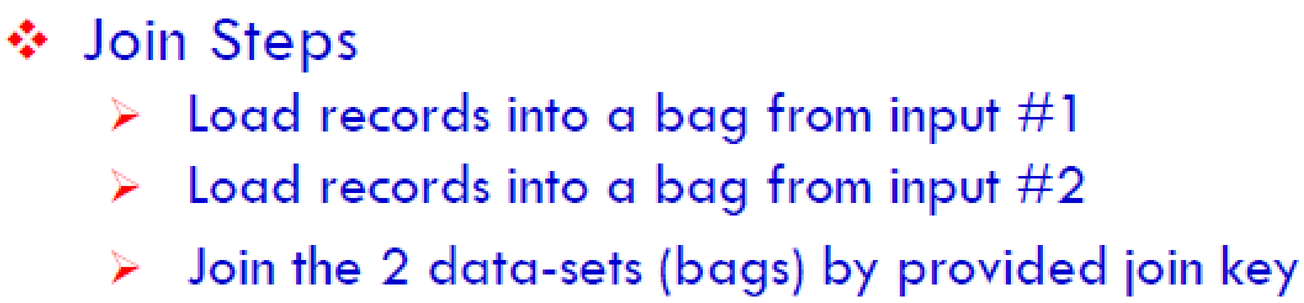
```
(Jim,35,Manager*Lead)
(John,30,Lead*Developer)
(Tom,35,Developer)
(Mary,30,Lead*Architect)
```

```
tokenBag = FOREACH role GENERATE name, TOKENIZE(role) as roletokens;
DUMP tokenBag;
```

```
(Jim,{(Manager),(Lead)})
(John,{(Lead),(Developer)})
(Tom,{(Developer)})
(Mary,{(Lead),(Architect)})
```

Joins

- The JOIN operation in Pig performs both inner and outer joins of two data sets using keys indicated for each input.

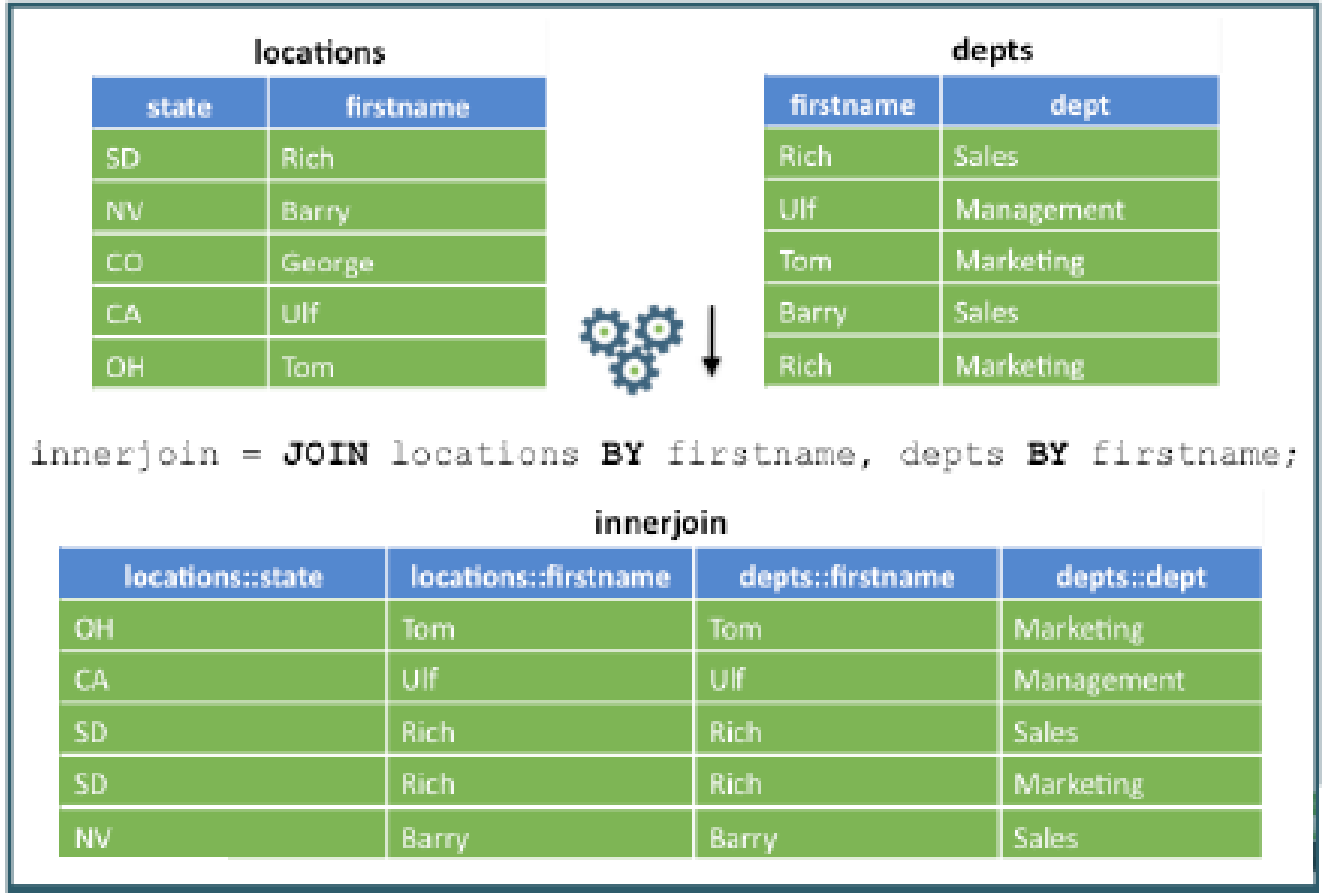


- Join Steps
  - Load records into a bag from input #1
  - Load records into a bag from input #2
  - Join the 2 data-sets (bags) by provided join key

Pig has the following joins

- Inner Join
- Outer Join

INNER Join



locations

state	firstname
SD	Rich
NV	Barry
CO	George
CA	Ulf
OH	Tom

depts

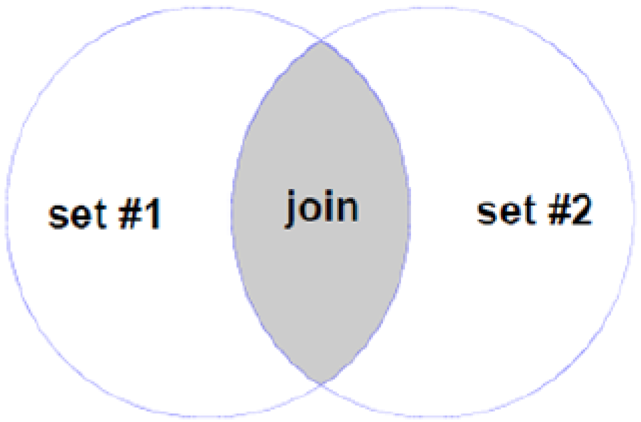
firstname	dept
Rich	Sales
Ulf	Management
Tom	Marketing
Barry	Sales
Rich	Marketing

```
innerjoin = JOIN locations BY firstname, depts BY firstname;
```

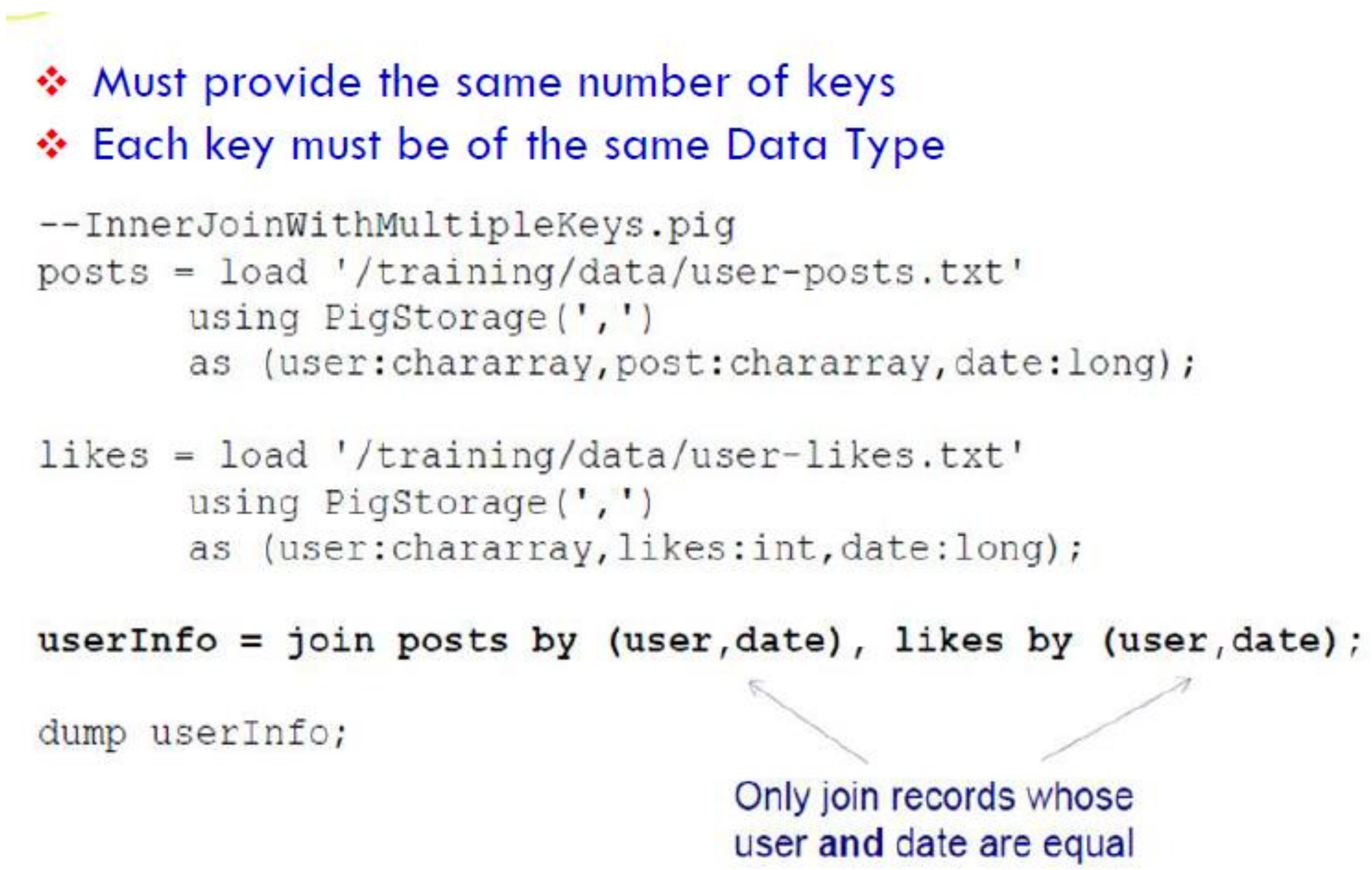
innerjoin

locations::state	locations::firstname	depts::firstname	depts::dept
OH	Tom	Tom	Marketing
CA	Ulf	Ulf	Management
SD	Rich	Rich	Sales
SD	Rich	Rich	Marketing
NV	Barry	Barry	Sales

Performing an Inner Join



Join by Multiple Keys



- Must provide the same number of keys
- Each key must be of the same Data Type

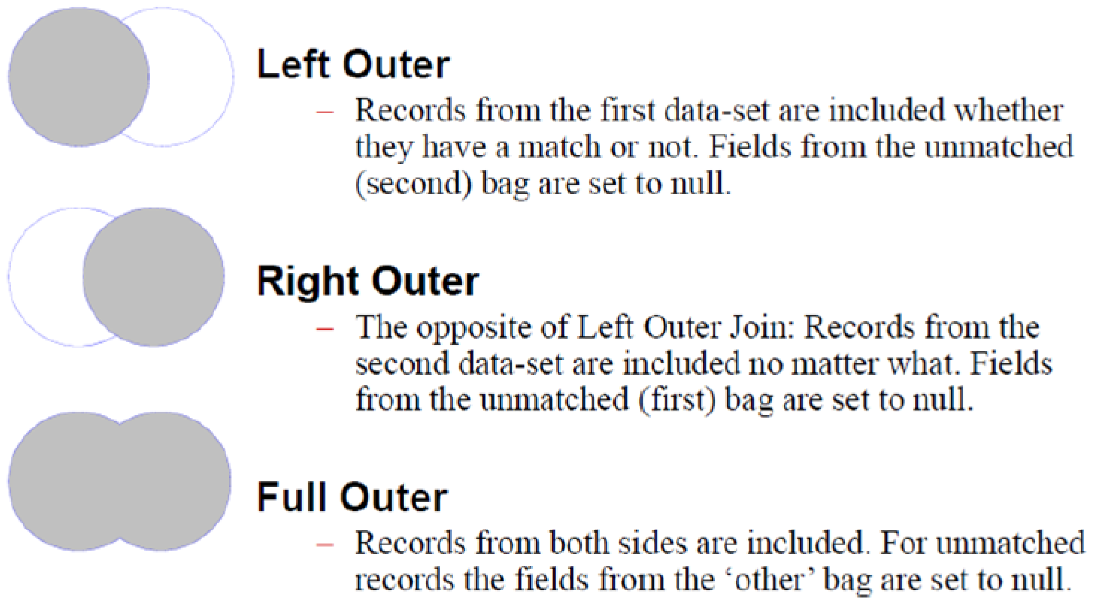
```
--InnerJoinWithMultipleKeys.pig
posts = load '/training/data/user-posts.txt'
        using PigStorage(',')
        as (user:chararray,post:chararray,date:long);

likes = load '/training/data/user-likes.txt'
        using PigStorage(',')
        as (user:chararray,likes:int,date:long);

userInfo = join posts by (user,date), likes by (user,date);

dump userInfo;
```

Only join records whose user and date are equal

OUTER Joins

- An outer join in Pig uses the OUTER keyword, along with either LEFT, RIGHT, or FULL.
- The syntax looks like:

alias = JOIN alias1 BY key1 [LEFT|RIGHT|FULL] OUTER, alias2 BY key2;



Left Outer

- Records from the first data-set are included whether they have a match or not. Fields from the unmatched (second) bag are set to null.

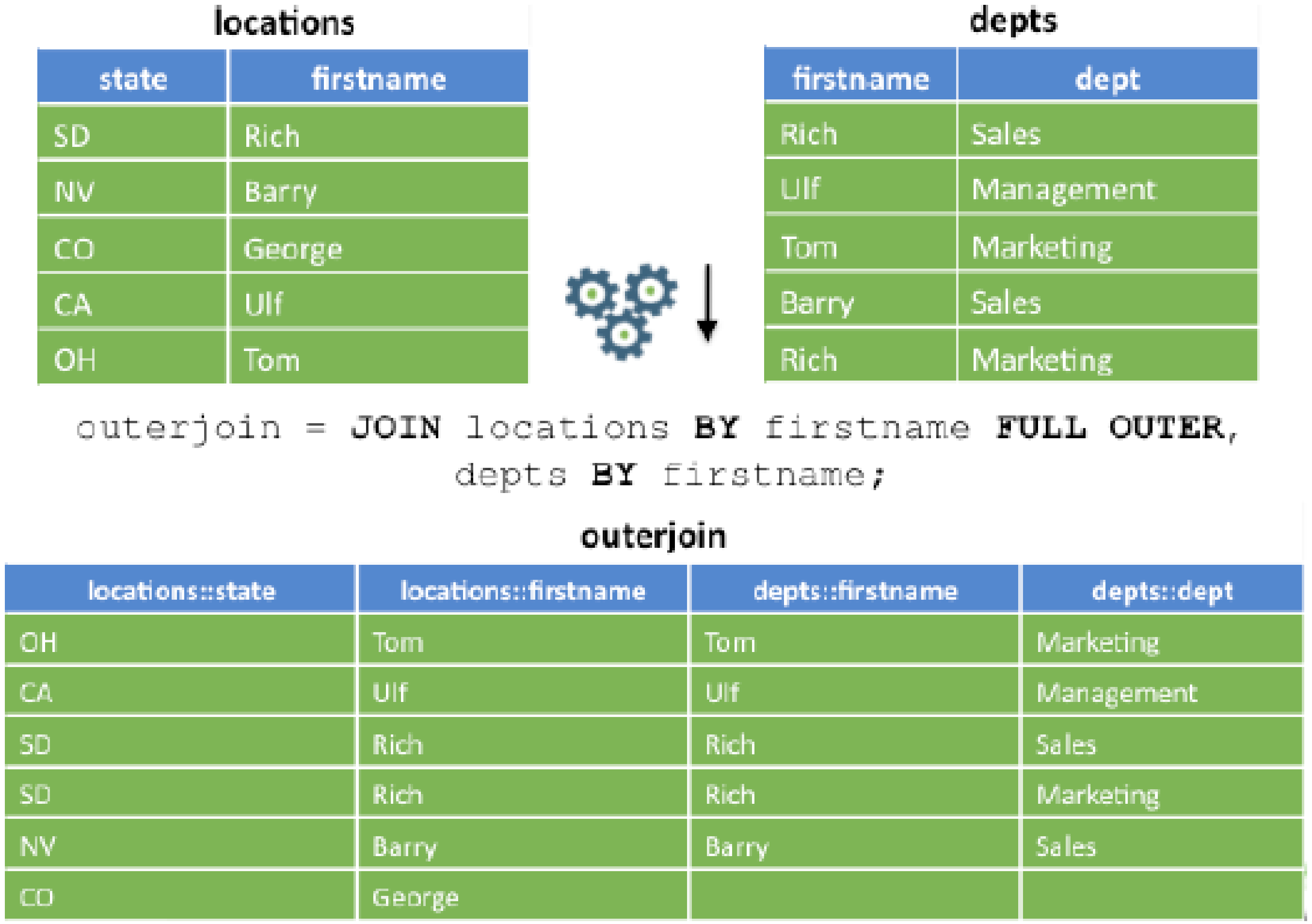
Right Outer

- The opposite of Left Outer Join: Records from the second data-set are included no matter what. Fields from the unmatched (first) bag are set to null.

Full Outer

- Records from both sides are included. For unmatched records the fields from the 'other' bag are set to null.

FULL OUTER Join



locations

state	firstname
SD	Rich
NV	Barry
CO	George
CA	Ulf
OH	Tom

depts

firstname	dept
Rich	Sales
Ulf	Management
Tom	Marketing
Barry	Sales
Rich	Marketing

```
outerjoin = JOIN locations BY firstname FULL OUTER,
            depts BY firstname;
```

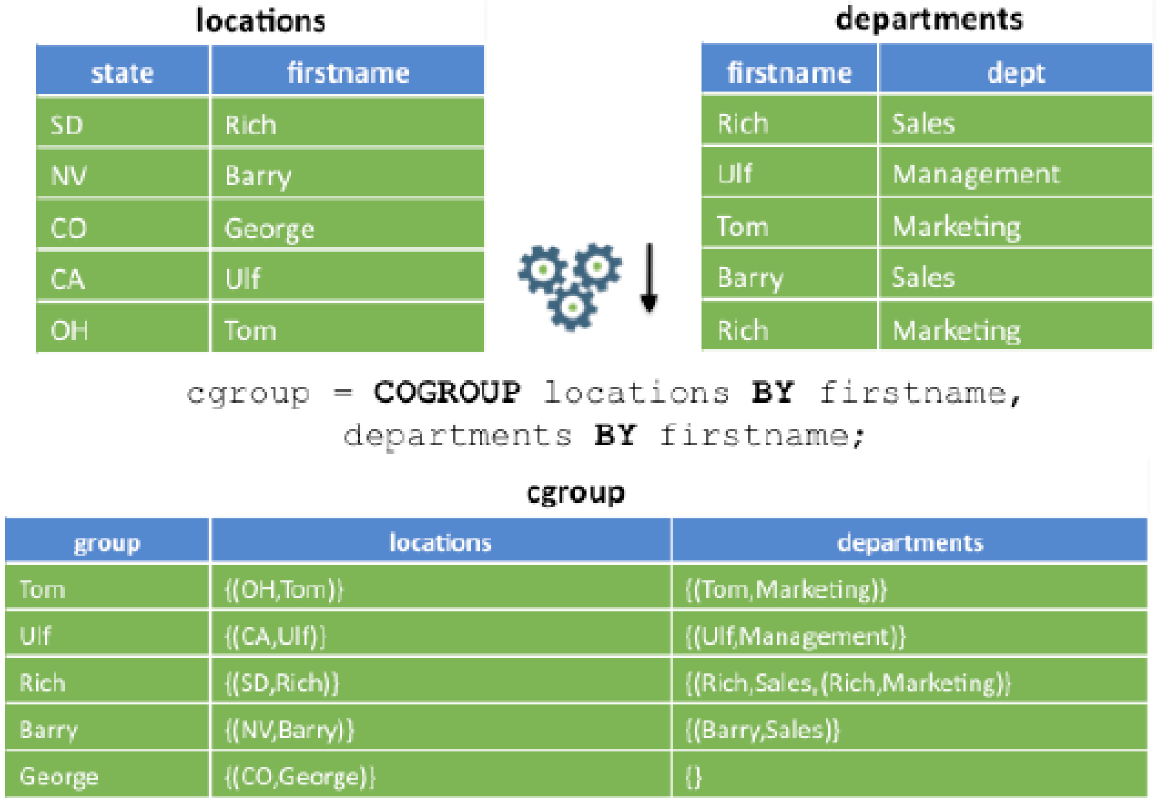
outerjoin

locations::state	locations::firstname	depts::firstname	depts::dept
OH	Tom	Tom	Marketing
CA	Ulf	Ulf	Management
SD	Rich	Rich	Sales
SD	Rich	Rich	Marketing
NV	Barry	Barry	Sales
CO	George		

Performing an Outer Join

The COGROUP Operator

- We use COGROUP when grouping together more than one relation
- For each input, the result of a COGROUP is a record with a key and one bag.



locations

state	firstname
SD	Rich
NV	Barry
CO	George
CA	Ulf
OH	Tom

departments

firstname	dept
Rich	Sales
Ulf	Management
Tom	Marketing
Barry	Sales
Rich	Marketing

```
cgroup = COGROUP locations BY firstname,
         departments BY firstname;
```

cgroup

group	locations	departments
Tom	{(OH,Tom)}	{(Tom,Marketing)}
Ulf	{(CA,Ulf)}	{(Ulf,Management)}
Rich	{(SD,Rich)}	{(Rich,Sales,(Rich,Marketing)}
Barry	{(NV,Barry)}	{(Barry,Sales)}
George	{(CO,George)}	{}

The COGROUP Operator

- Cogroup by default is an OUTER JOIN
- You can remove empty records with empty bags by performing INNER on each bag

Ex:

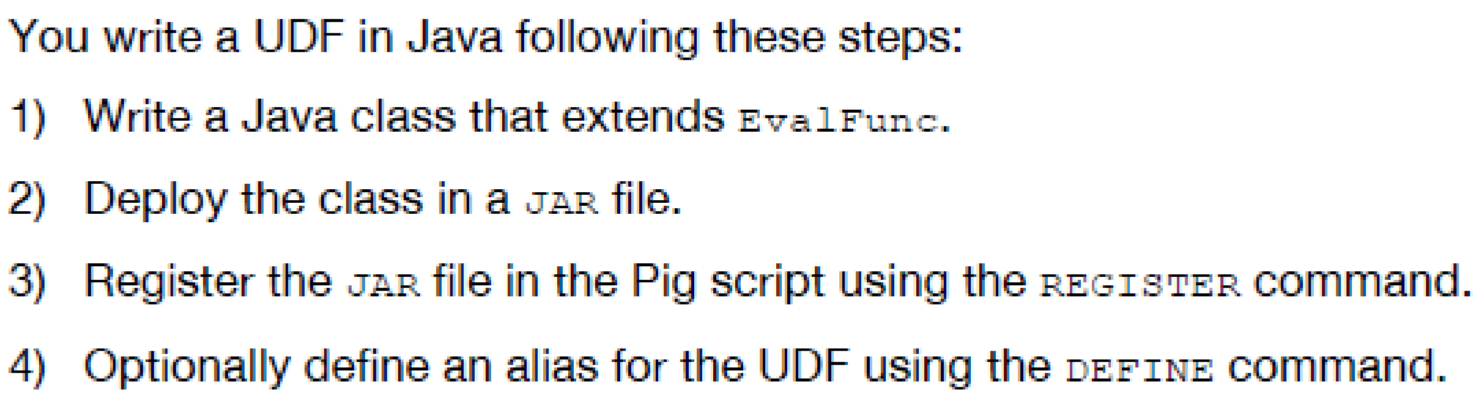
COGROUP locations BY firstname INNER, departments BY firstname ;

Pig UDF

- Pig provides extensive support for user defined functions (UDFs) as a way to specify custom processing.
- Pig (UDFs) can be written in six languages:

Java,Jython,Python,Jruby,JavaScript,Groovy

- The UDF class extends the EvalFunc class which is the base for all Eval functions.



You write a UDF in Java following these steps:

1) Write a Java class that extends `EvalFunc`.
2) Deploy the class in a `JAR` file.
3) Register the `JAR` file in the Pig script using the `REGISTER` command.
4) Optionally define an alias for the UDF using the `DEFINE` command.

Invoking UDF

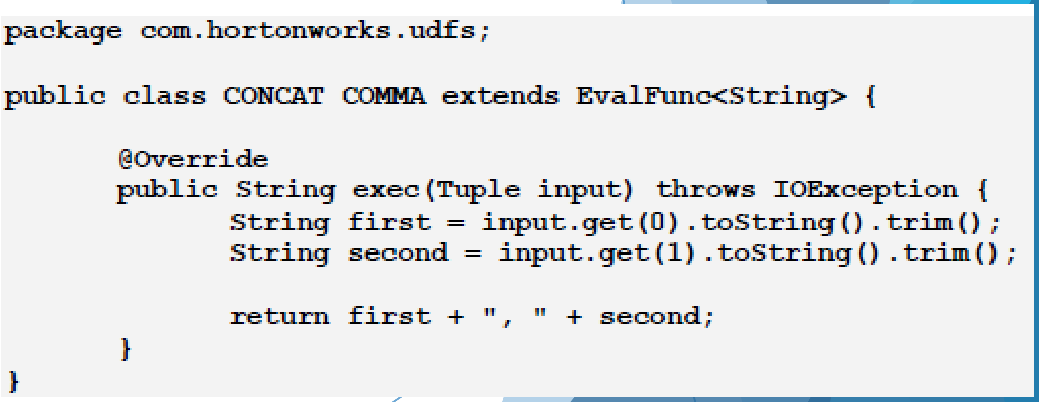
- For invoking UDF, the function needs to be registered by your Pig script so that the Pig compiler knows where to find the definition of the UDF.

1) Use the REGISTER command to register a JAR:

register my.jar;

2) As an option, you can use the DEFINE command to define an alias that simplifies the syntax for invoking the UDF:

DEFINE CONCAT_COMMA com.hortonworks.udfs.CONCAT_COMMA();



```
package com.hortonworks.udfs;

public class CONCAT COMMA extends EvalFunc<String> {

        @Override
        public String exec(Tuple input) throws IOException {
                String first = input.get(0).toString().trim();
                String second = input.get(1).toString().trim();

                return first + ", " + second;
        }
}
```

3) Now you can invoke the UDF using the alias:

x = FOREACH logevents GENERATE CONCAT_COMMA(level, code);

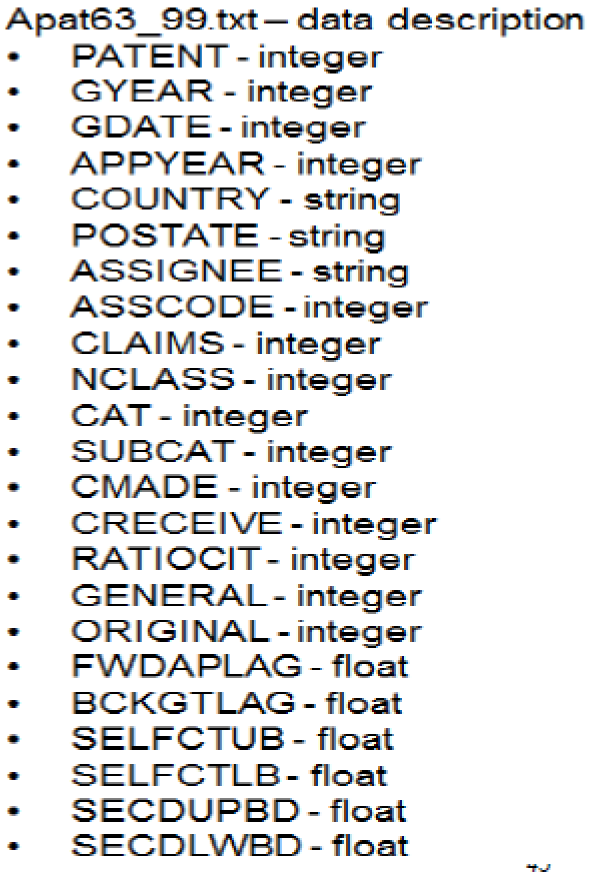
Or

x = FOREACH logevents GENERATE com.hortonworks.udfs.CONCAT_COMMA(level, code);

Hands On

Pig Hands On

- Source File
  - File Name: apat63_99.txt
  - File Content: PatentID, Year, Applicable Year, Country, State …



Apat63_99.txt – data description

- PATENT - integer
- GYEAR - integer
- GDATE - integer
- APPYEAR - integer
- COUNTRY - string
- POSTATE - string
- ASSIGNEE - string
- ASSCODE - integer
- CLAIMS - integer
- NCLASS - integer
- CAT - integer
- SUBCAT - integer
- CMADE - integer
- CRECEIVE - integer
- RATIOCIT - integer
- GENERAL - integer
- ORIGINAL - integer
- FWDAPLAG - float
- BCKGTLAG - float
- SELFCTUB - float
- SELFCTLB - float
- SECDUPBD - float
- SECDLWBD - float

- Copy the file to HDFS

- Task 1
  - Year and Country wise Patent Count

- Task 2
  - Select Top 5 Country's Patent Count

- Task 3
  - State wise Patent Count for US

Thank You

Keerthiga Barathan