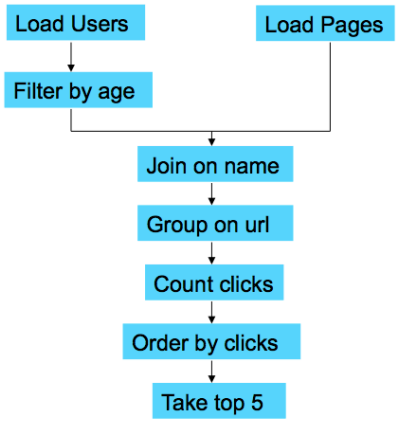
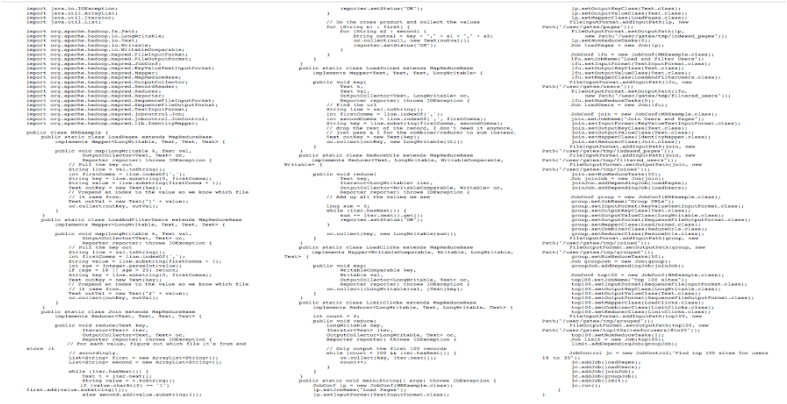
PIG LATIN

Pig Motivation

- Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited sites by users aged 18 - 25.



MR Solution



170 lines of code, 4 hours to write

3

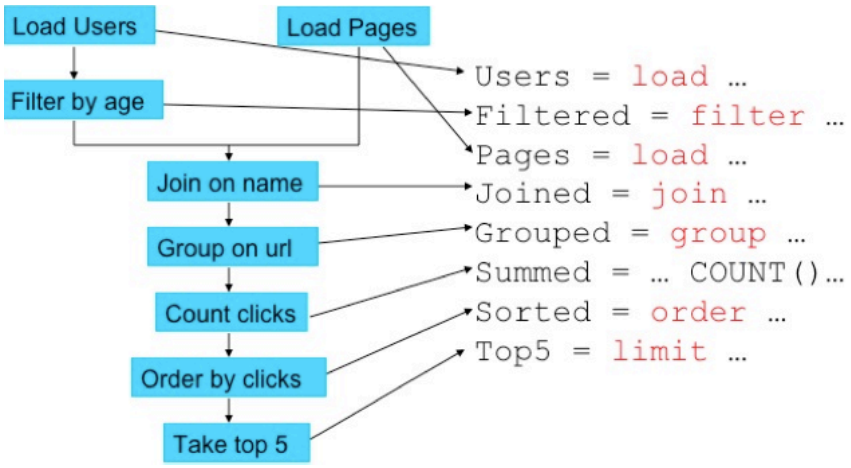
Pig Latin Solution

```
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';
```

9 lines of code, 15 minutes to write

4

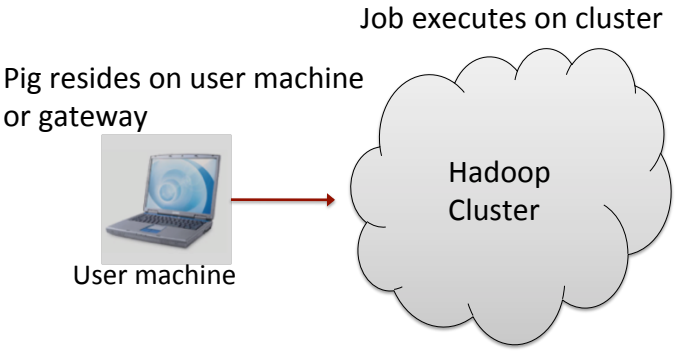
Natural Fit With Dataflow



5

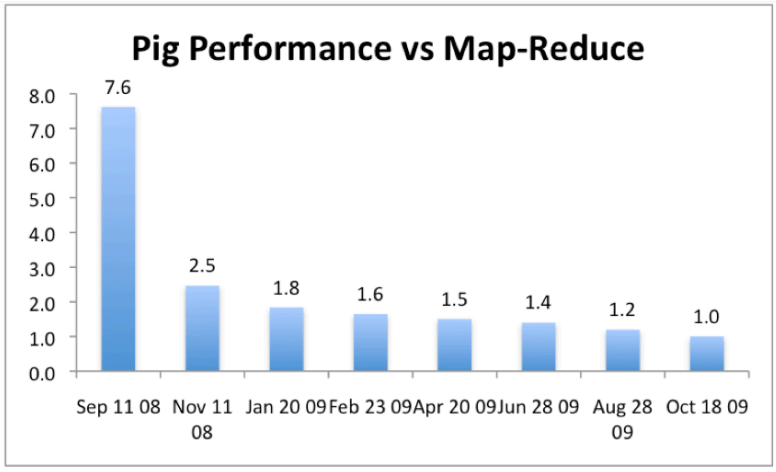
Pig Deployment

No server, all optimization and planning done on the launching machine



6

Pig Latin Performance



7

Pig-Latin Overview

- Data model = loosely typed *nested relations*
- Query model = Relational Algebra (less SQL)
- Execution model:
  - Option 1: run locally on your machine
  - Option 2: run on AWS, compiles into MR

8

SQL Example

```
SELECT category, AVG(pagerank)
FROM Pages
WHERE pagerank > 0.2
GROUP By category
HAVING COUNT(*) > 10^6
```

1. Read pages
2. Filter by pagerank
3. Group by category
4. Filter by page count
5. Output avgepagerank

```
Pages(url, category, pagerank)
```

9

SQL Example

1. Read pages
2. Filter by pagerank
3. Group by category
4. Filter by page count
5. Output avge pagerank

```
pages =
   LOAD 'pages-file.txt' as
              (url, category, pagerank);
good_pages = FILTER pages BY pagerank > 0.2;
groups = GROUP good_pages BY category;
big_groups = FILTER groups
     BY COUNT(good_pages) > 10^6;
output = FOREACH big_groups
     GENERATE category,
              AVG(good_urls.pagerank);
```

```
Pages(url, category, pagerank)
```

10

Pig vs SQL

SQL

- Declarative
- Push evaluation
- Flat data model
- Single monolithic expression
- Non-compositional
  - E.g. GROUP BY… is not a table
- JOIN is primitive

Pig

- Procedural (dataflow)
- Pull (lazy) evaluation
- Nested relations
- Step-by-step combination of data transformations
- Compositional
  - E.g. GROUP BY … tuple of sets
- JOIN is defined

11

Simple Data Types

- *int* : 42
- *long* : 42L
- *float* : 3.1415f
- *double* : 2.7182818
- *chararray* : UTF-8 String
- *bytearray* : blob

12

Complex Types

- Tuple: Ordered set of fields
  - Field can be simple or complex type
  - `('Alice', 55, 'salesperson')`
- Bag: Collection of tuples
  - Can contain duplicates
  - `{('Alice','sales'), ('Betty','finance'), ...}`
- Map: Set of (key, value) pairs

13

Complex Types

- Tuple components can be referenced by name or by number
  - `$0, $1, $2, …`
  - `url, category, pagerank,...`
- Bags can be nested !
  - ```
    {('a', {1,4,3}),
     ('c',{ }),
     ('d', {2,2,5,3,2})}
    ```

14
    ```

Pig Expressions

$$t = \left(\text{'alice'}, \left\{\begin{array}{c}(\text{'lakers'}, 1)\\(\text{'iPod'}, 2)\end{array}\right\}, [\text{'age'} \rightarrow 20]\right)$$

Let fields of tuple t be called f1, f2, f3

Expression Type	Example	Value for t
Constant	'bob'	Independent of t
Field by position	$0	'alice'
Field by name	f3	'age' → 20
Projection	f2.$0	{('lakers') ('iPod')}
Map Lookup	f3#'age'	20
Function Evaluation	SUM(f2.$1)	1 + 2 = 3
Conditional Expression	f3#'age'>18? 'adult':'minor'	'adult'
Flattening	FLATTEN(f2)	'lakers', 1 'iPod', 2

15

Loading Data using PigStorage

```
query = load '/user/piguser/query.txt' using PigStorage()
       as (userId, queryString, timestamp)

queries = load '/user/pigusers/querydir/part-*' using
       PigStorage() as (userId, queryString, timestamp)

queries = load 'query_log.txt' using PigStorage('\u0001') as
       (userId, queryString, timestamp)
```

All files under querydir containing part-* are loaded

- userID, queryString, timestamp fields tab separated
- PigStorage('{delimiter}')
  - Loads records with fields delimited by {delimiter}
  - Unicode representation '\u0001' for Ctrl+A,
  - Default is TAB

16

Storing data using PigStorage and viewing Data

```
store joined_result into '/user/piguser/output' ;
store joined_result into '/user/piguser/myoutput/'
       using PigStorage('\u0001');
store joined_result into '/user/piguser/myoutputbin'
       using BinStorage();
store sports_views into '/user/piguser/myoutput' using PigDump();

dump sports_views;
```

- Default PigStorage
- PigStorage ('{delimiter}')
  - BinStorage store arbitrarily nested data
  - Used for intermediate results
- Dump displays data on terminal

```
(alice,lakers,3L)
(alice,lakers, 0.7f)
```

17

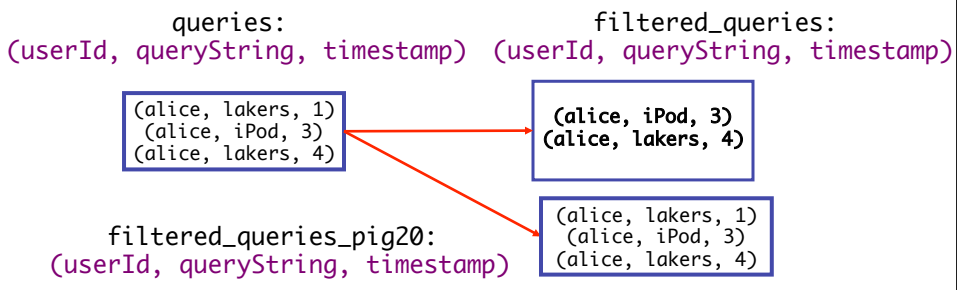
Filtering Data

```
queries = load 'query_log.txt' using PigStorage('\u0001') as
    (userId: chararray, queryString: chararray, timestamp: int);

filtered_queries =  filter queries by timestamp > 1;

filtered_queries_pig20 = filter queries by timestamp is not null;

store filtered_queries into  'myoutput' using PigStorage();
```



18

(Co)Grouping Data

```
Data 1:
queries: (userId: chararray,
          queryString : chararray,
          timestamp: int)

Data 2:
sports_views: (userId: chararray,
               team: chararray,
               timestamp: int)
```

```
queries = load query.txt as ()..;
sport_views = load sports_views.txt as ….;

team matches = cogroup  sports_views by (userId, team),
                        queries by (userId, queryString);
```

19

Example of Cogrouping

sports_views:
(userId, team, timestamp)

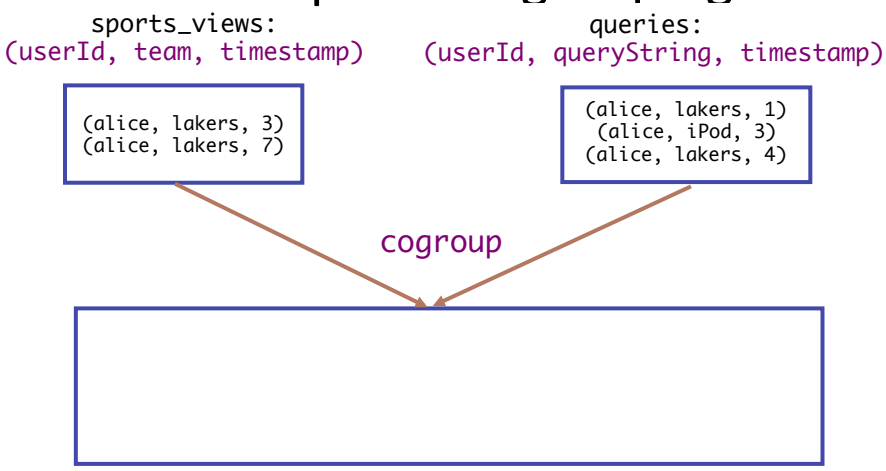
```
(alice, lakers, 3)
(alice, lakers, 7)
```

queries:
(userId, queryString, timestamp)

```
(alice, lakers, 1)
(alice, iPod, 3)
(alice, lakers, 4)
```

20

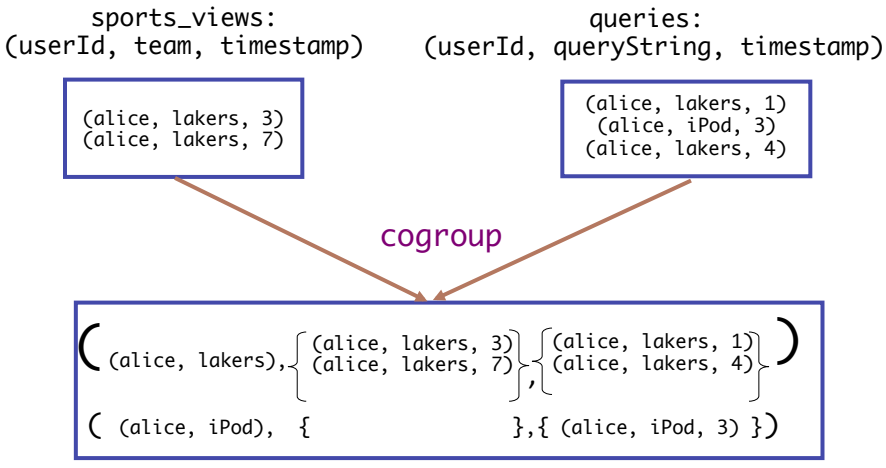
Example of Cogrouping




```
team matches = cogroup  sports_views by (userId, team),
                        queries by (userId, queryString);
```

21

Example of Cogrouping

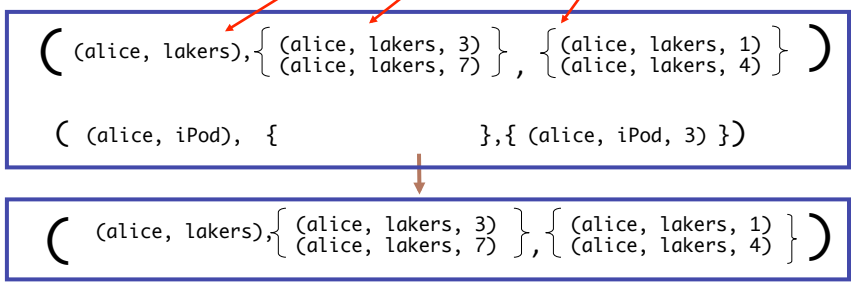


22

Filtering records of Grouped Data

```
filtered_matches = filter team_matches
                   by COUNT(sports_views) > '0';
```

team_matches: (group, sports_views, queries)

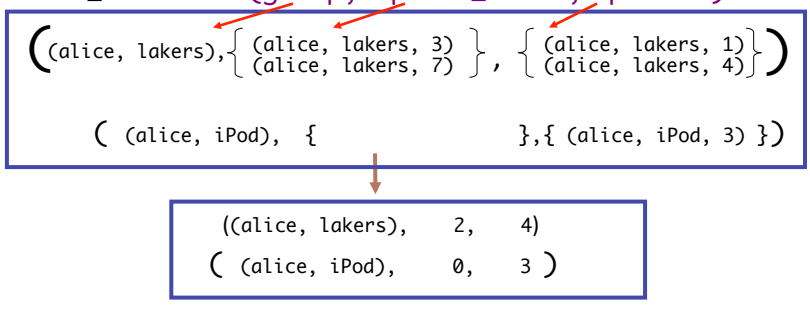


23

FOREACH: Per-Record Transformations

```
times_and_counts = foreach team_matches
    generate group, COUNT(sports_views), MAX(queries.timestamp);
```

team_matches: (group, sports_views, queries)



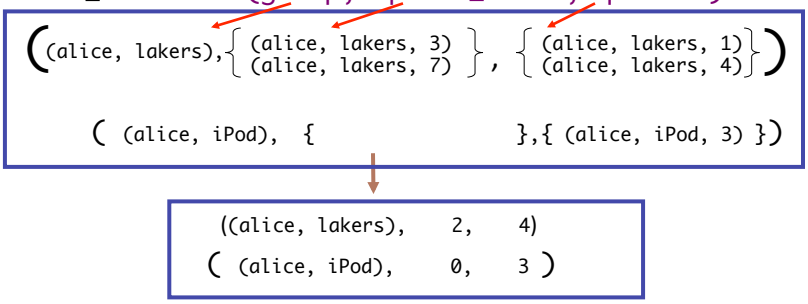
times_and_counts: (group, -, -)

24

FOREACH: Per-Record Transformations

```
times_and_counts = foreach team_matches
    generate group, COUNT(sports_views) as count,
                    MAX(queries.timestamp) as max;
```

team_matches: (group, sports_views, queries)

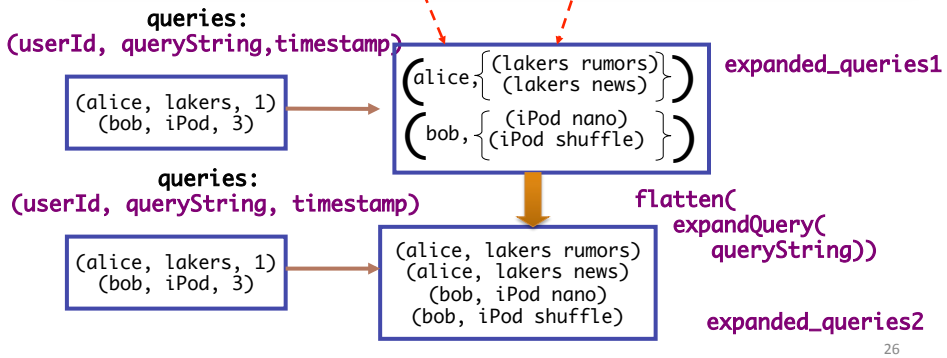


times_and_counts: (group, count, max)

Eliminating Nesting: FLATTEN

```
expanded_queries1 = foreach queries
            generate userId, expandQuery(queryString);

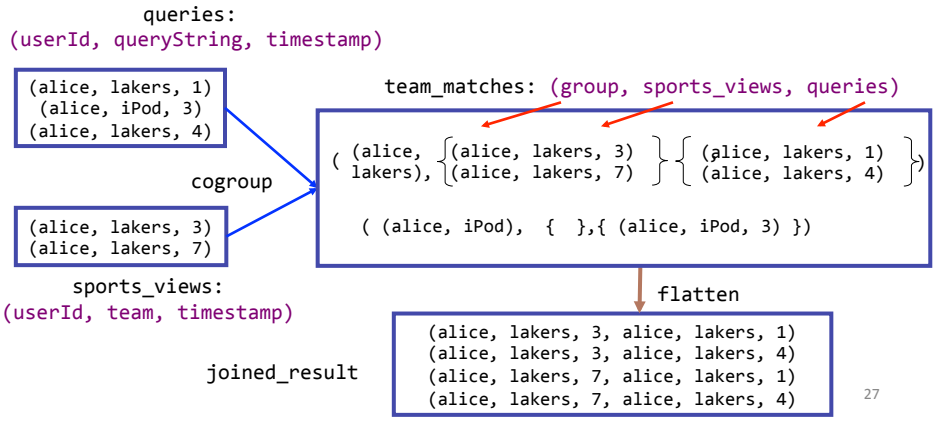
expanded_queries2 =  foreach queries
            generate userId, flatten(expandQuery(queryString));
```



Flattening Multiple Items

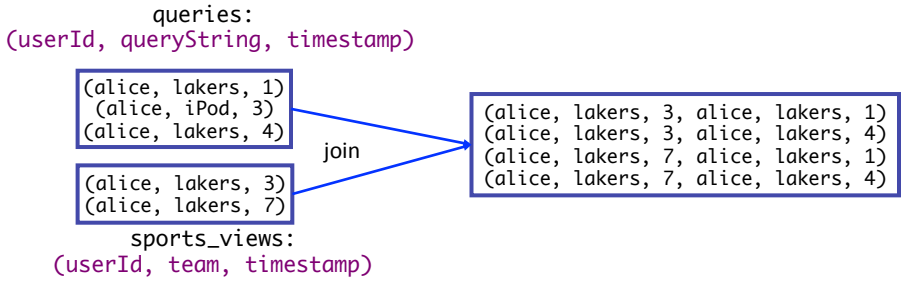
```
team_matches = cogroup  sports_views by (userId, team),
                        queries by (userId, queryString);

joined_result = foreach team_matches generate
                    flatten(sports_views),flatten(queries);
```

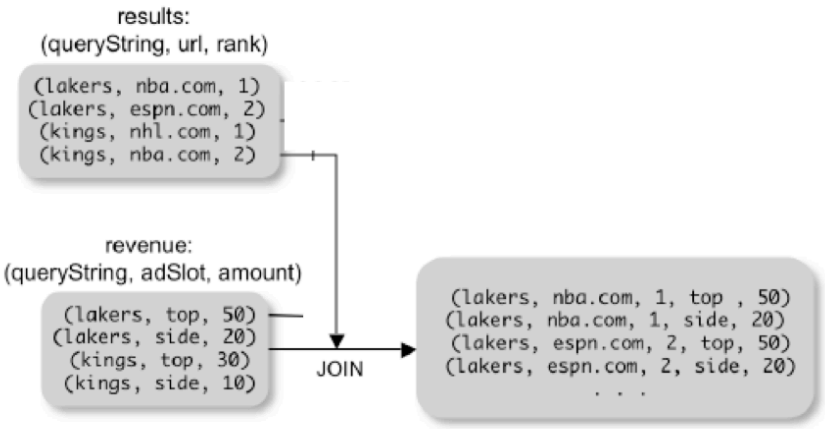


JOIN = COGROUP + FLATTEN

```
joined_result = join sports_views by (userId, team),
                queries by (userId, queryString);
```

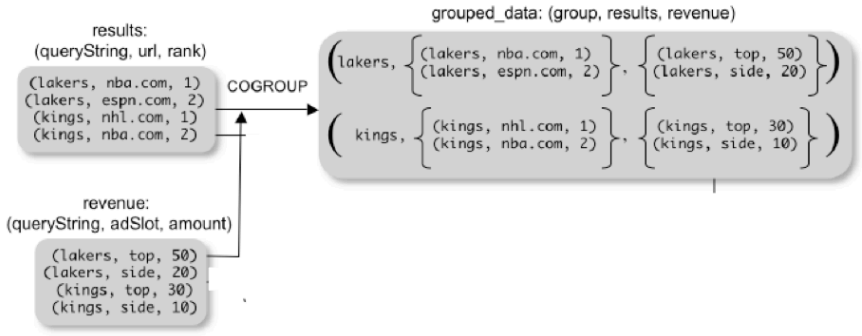


Join



29

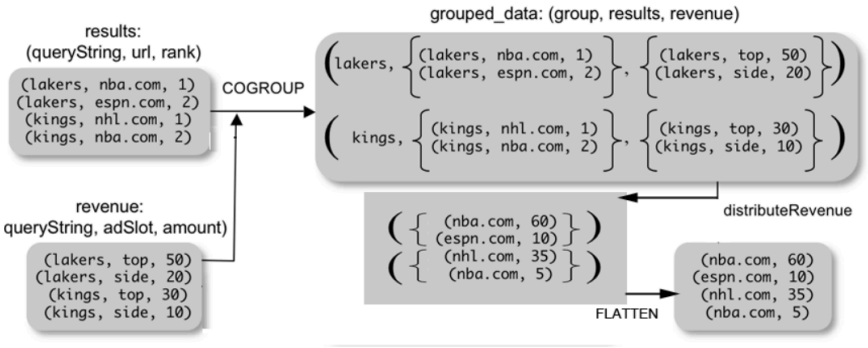
Co-Group



30

Cogroup vs Join

```
url_revenues =
      FOREACH grouped_data GENERATE
      FLATTEN(distributeRev(results, revenue));
```



31

Cogroup vs Join

- Why COGROUP and not JOIN?
  - May want to process nested bags of tuples before taking the cross product.
  - Keeps to the goal of a single high-level data transformation per Pig Latin statement

```
JOIN results BY queryString,
     revenue BY queryString;
```

Equivalent

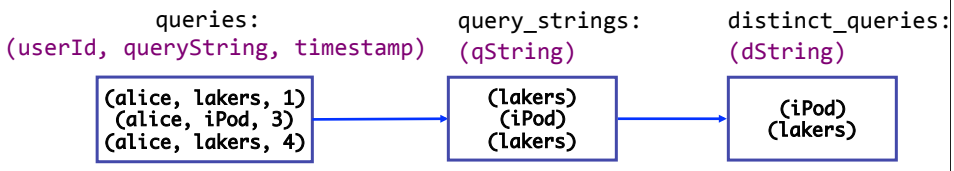
```
temp = COGROUP results BY queryString,
               revenue BY queryString;
join_result = FOREACH temp GENERATE
              FLATTEN(results), FLATTEN(revenue);
```

Eliminating duplicates using distinct

```
queries = load 'queries.txt'
          as (userId, queryString: chararray, timestamp);

query_strings = foreach queries generate queryString as qString;

distinct_queries = distinct query_strings as dString;
```



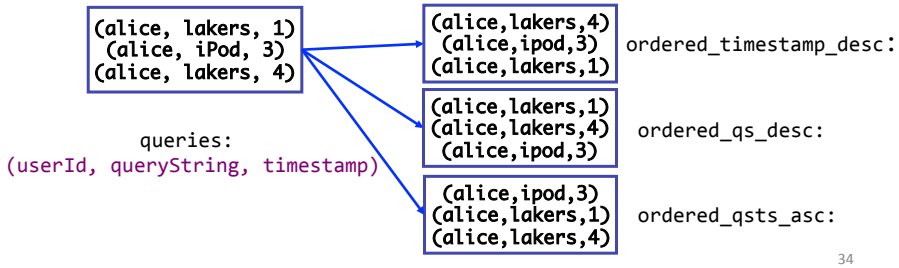
33

Ordering Data

```
queries = load 'queries.txt'
  as (userId: chararray, queryString: chararray, timestamp: int);
-- Pig 2.0

ordered_timestamp_desc = order queries by timestamp desc;
ordered_qs_desc = order queries by queryString desc;

ordered_qsts_asc = order queries by queryString, timestamp
                   parallel 7;
```

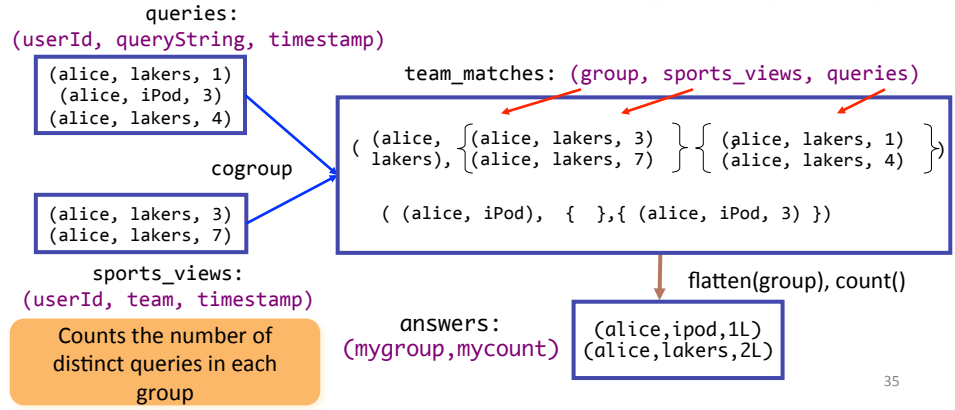


34

Nested Operations

```
team matches = cogroup  sports_views by (userId, team),
                        queries by (userId, queryString);

answers = foreach team_matches { distinct_qs = distinct queries;
                      generate flatten(group) as mygroup,
                               COUNT(distinct_qs) as mycount; }
```



Counts the number of distinct queries in each group

35

Pig Unigrams

- Input: Large text document
- Process:
  - Load the file
  - For each line, generate word tokens
  - Group by word
  - Count words in each group

36

Load

```
myinput = load '/user/piguser/text.txt'
     USING TextLoader()
     AS (myword:chararray);

{
     (program program)
     (pig pig)
     (program pig)
     (hadoop pig)
     (latin latin)
     (pig latin)
}
```

37

Tokenize

```
words = FOREACH myinput
     GENERATE FLATTEN(TOKENIZE(*));

{
     (program) (program)
     (pig) (pig)
     (program) (pig)
     (hadoop) (pig)
     (latin) (latin)
     (pig) (latin)
}
```

38

Group

```
grouped = GROUP words BY $0;

{
     (pig, {(pig),(pig),(pig),(pig),(pig)})
     (latin, {(latin),(latin),(latin)})
     (hadoop, {(hadoop)})
     (program, {(program),(program),
               (program)})
}
```

39

Count

```
counts = FOREACH grouped
     GENERATE group, COUNT(words);

{
     (pig, 5L)
     (latin, 3L)
     (hadoop, 1L)
     (program, 3L)
}
```

40

Store

```
store counts
     into '/user/piguser/output'
     using PigStorage();
```

```
pig 5
latin 3
hadoop 1
program 3
```

41

Simple Map-Reduce

```
-- input : {(field1, field2, field3, . . .)}

map_result = FOREACH input
               GENERATE FLATTEN(map_op(*))
map_result : {(a1, a2, a3, . . .)}

key_groups = GROUP map_result BY $0
key_groups : {(a1, {(a2, a3, . . .)})}

output = FOREACH key_groups
               GENERATE reduce_op($1)
```

42

Conclusions

- Pig Latin: Domain specific language for cloud computing
- Building on Hadoop/MR
- Building on HDFS
- Building on Hbase (TBC)