declarative languages

Large Scale Data Processing

Adaptation from

Magdalena Balazinska (Univ. of Washington)
Mining of Massive Datasets, by Rajaraman and Ullman
Alan Gates (Yahoo!)
Olston

Declarative Languages

- Map-Reduce framework hides scheduling and parallelization details
- Limited query expressiveness
  - Complex queries difficult to write
- Declarative languages on top of map-reduce
  - Pig Latin (Yahoo!)
    - Like relational algebra
    - Open source
  - HiveQL (Facebook)
    - SQL like language
    - Open source
  - SQL / Tenzing
    - proprietary

Pig (Latin)

- Pig Latin   english based
  - A higher SQL like language to run complex queries that require several map-reduce jobs
- Pig
  - An execution engine
    - Translates Pig Latin programs into graphs of map-reduce jobs
    - Executes them on top of Hadoop
    - An Apache open source project

Example (Alan Gates, Yahoo)

users(name, age), pagelog(url, uname)

Find the top 5 most popular pages for users aged 18-25

SQL:

```
SELECT  url, count(*) as clicks
FROM users U, pagelog P
WHERE U.name = P.uname
AND U.age >= 18
AND U.age <= 25
GROUP BY URL
ORDER BY clicks desc
LIMIT 5                                  -- FETCH FIRST 5 ROWS ONLY
```

Map reduce pogram: 170 lines of code

In Pig Latin

red = key words
blue = file/input

Users = **load** *'users'* **as** (name,age); → load file into variable

Fltrd = **filter** Users **by** age >= 18 **and** age <= 25; selection

Pages = **load** '*pages*' **as** (uname, url);

Jnd = **join** Fltrd **by** name, Pages **by** uname;

Grpd = **group** Jnd **by** url;

Smmd = **foreach** Grpd **generate** ($0), **COUNT**($1) **as** clicks;

Srtd = **order** Smmd **by** clicks **desc**;

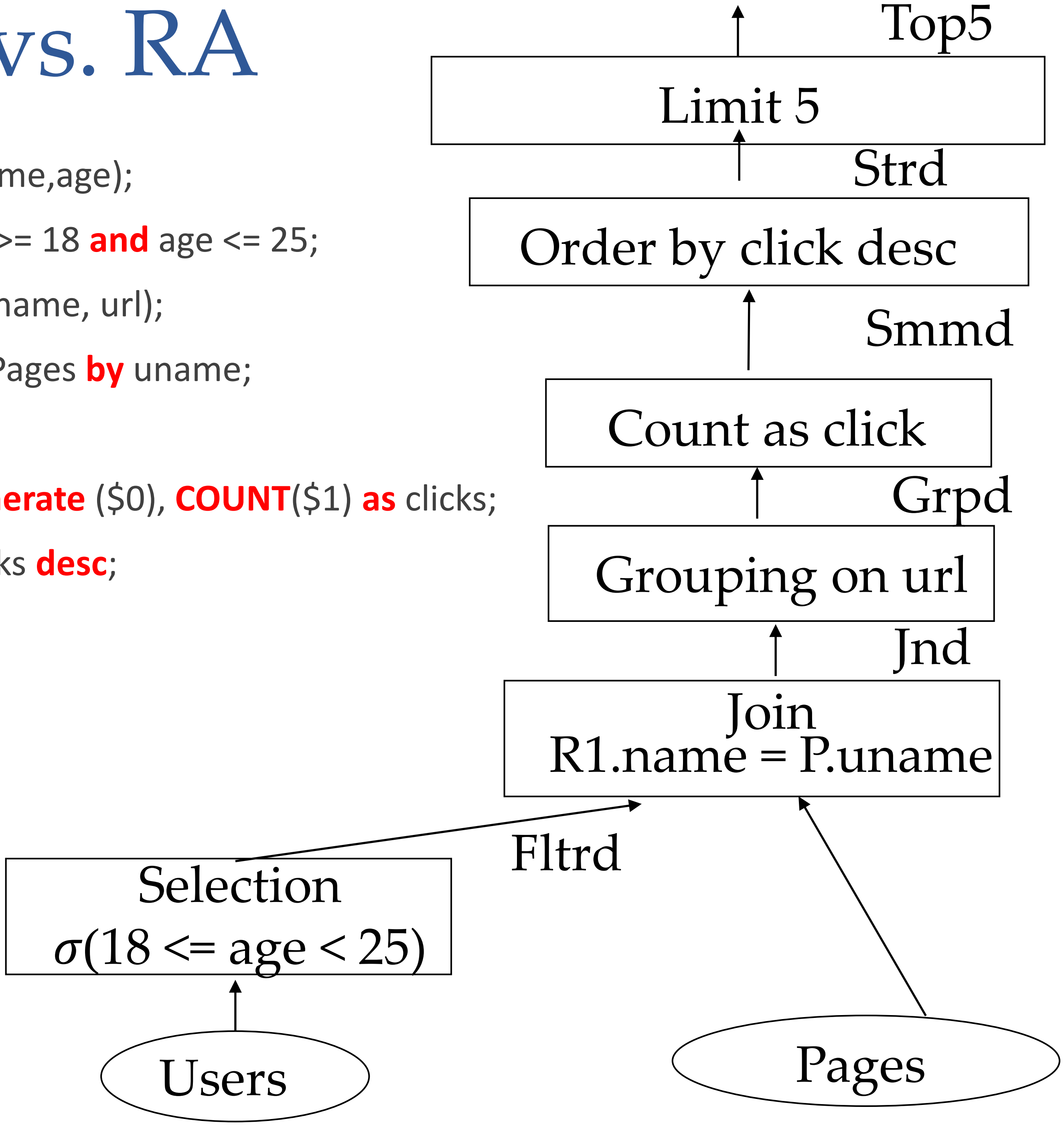
Top5 = **limit** Srtd 5;

store Top5 **into** '*top5sites*'

Very similar to Relational Algebra

Pig Latin vs. RA

```
Users = load 'users' as (name,age);
Fltrd = filter Users by age >= 18 and age <= 25;
Pages = load 'pages' as (uname, url);
Jnd = join Fltrd by name, Pages by uname;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate ($0), COUNT($1) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites'
```



Load and Store

- Users = **load** *'users'* **as** (name,age);
- **Load:** read information from file into a (temp) relation
  - Mostly user defined to translate file format into a relational format
- **Store**: write relation into file
  - Again, usually provided by user

Pig Latin Operators

- Fltrd = **filter** Users **by** age >= 18 **and** age <= 25;
  - Left side: new intermediate relation
  - Right side operation on existing relations
- Operators
  - Selection
    - Res **= filter** R1 **by:**
    - SELECT * FROM R1 WHERE …
    - By age >= 18; by url matches ‘*oracle*'
  - Join
    - Res **= join** R1 **by** a1, R2 **by** a2:
    - SELECT * FROM R1, R2 WHERE R1.a1 = R2.a2
  - Order by
    - Res **= order** R1 **by** a1 **desc**
    - SELECT * FROM R1 order by a1 desc

Group By

- Given relation Rel(A, B, C) with three tuples

  (a1, b1, c1)

  (a1, b2, c2)

  (a3, b3, c3)

- Grpd = **group** Rel **by** A;
- Result relation is Grpd(group, Rel)
  - Attribute 'group' has the same type as attribute A of Rel
  - Attribute 'Rel' is a multiset (bag)
  - In the given example, Grpd has two tuples, one for each value of A; first attribute of the tuple is the value of A, the second is the set of all tuples of Rel that have this particular value of A
  - dump Grpd:
    - (a1, {(a1,b1,c1), (a1,b2,c2)})
    - (a3, {(a3, b3, c3)}

For each

- Assume same as before
  - Grpd = **group** Rel **by** A;
  - Result relation is Grpd(group, Rel):
    - (a1, {(a1,b1,c1), (a1,b2,c2)})
    - (a3, {(a3, b3, c3)})
- For each (two options)
  - Smmd = **foreach** Grpd **generate** ($0), **COUNT**($1) **as** c;
  - Smmd = **foreach** Grpd **generate** group, **COUNT**(Rel) **as** c;
- Result relation is Smmd(group, c)
  - Attribute 'group' has the same type as attribute A of Rel
  - Attribute c a long
  - dump Smmd: to screen
    - (a1, 2L) 2 elements
    - (a3, 1L) 1 element

Projection and others

- Projection
  - Assume R1(A, B, C)
  - Rel = **for each** R1 **generate** A, B;

Data Model and Flattening

- Supported types: big data supports
  - Atomic (string, number…)
  - Tuple (58, ‘lilly’, 10, 10)
  - Multiset {(58, ‘lilly’, 10, 10), (33, ‘debby’, 5, 7)}
  - Further nesting possible: (1, (2,3))
  - Maps (advanced)
- Flattening example
  - Assume R = {(1, (2,3))}
  - Res = foreach R generate $0, flatten($1)

    make it appear as 1 list instead of nested
  - Res = {(1, 2, 3)}
  - Semantics somewhat obscure…
- Sometimes output type not quite clear: try to flatten…

Implementation

- Parser and Query Generator:
  - Everything between load and store translates into one logical plan
- Logical plan:
  - Graph of Hadoop map-reduce jobs
- All statements between two groups → one Map-reduce job

doesn't execute anything
before dump or store

Map Reduce Assignment

- Use Pig Latin on top of Hadoop to process (not so) large data set
- We have set up a hadoop cluster with 4 nodes (virtual)
- You have access to that cluster
- You have to write PigLatin Queries
- You have to observe the execution
- Instructions will be on myCourses