ISIT312 Big Data Management

Pig Latin

Dr Fenghui Ren

School of Computing and Information Technology -
University of Wollongong

Pig Latin

Outline

Preliminary Matters

Pig Latin is a dataflow language

Each processing step results in a new data set, or relation

Loading, filtering, and transforming

```
A = load 'NYSE_dividends' (exchange, symbol, date, dividends);
B = filter A by dividends > 0;
C = foreach B generate UPPER(symbol);
```

Case Sensitivity

- `A = load 'foo';` is not equivalent to `a = load 'foo';`

Comments

Comments

```
A = load 'foo'; --this is a single-line comment
/*
  This is a multiline comment.
*/
```

Pig Latin

Outline

Input in Pig Latin

`load`

By default, load looks for your data on HDFS in a tab-delimited file using the default load function `PigStorage`

Loading data
```
divs = load '/data/examples/file';
```

A statement above looks for a file called 'file' in a folder `hdfs://[host-name]:[port- name]/data/examples`

It is possible to determine a particular load function

Loading data HBaseStorage load function
```
divs = load '/data/examples/file' using HBaseStorage(,);
```

It is also possible to specify the schema when loading the file

Loading with a schema
```
divs = load '/data/examples/file' as (exchange: int, symbol: chararray);
```

Output in Pig Latin

`store`

Pig stores your data on HDFS in a tab-delimited file using PigStorage

Storing data

```
store processed into '/data/examples/processed';
```

As in `load`, you can also specify a store function, e.g. `HbaseStorage()`

In `processed`, there are usually multiple part files rather than a single file (why ?)

`dump`

Occasionally you will want to see it on the screen, this is done by the `dump` action

Displaying data

```
dump processed;
```

Pig Latin

Outline

Arithmetic, Boolean and relational Operations

Pig Latin supports the following basic operations

- arithmetic operations: `+`, `-`, `*`, `/`
- Boolean operations: `and`, `or`, `not`
- relational operations: `==`, `!=`, `<`,`>`, `<=`, `>=`, `matches` (pattern matching)

Relational operations are the main tools Pig Latin provides to operate on your data

They are the horsepower of Pig Latin

They allow you to transform the operations such as sorting, grouping, joining, projecting, and filtering

Pig Latin

Outline

Built-In Functions

Mathematical functions

- `AVG`, `COUNT`, `MAX`, `MIN`, `SIZE`, `SUM`, `ABS`, `CEIL`, `EXP`, `FLOOR`,`LOG`, `RANDOM`, `ROUND`, and others

String functions

- `STARTWITH`, `ENDWITH`, `LOWER`, `UPPER`, `REGEX_EXTRACT`, `TOKENIZE`, and others

Date/time functions

- `CurrentTime`, `DaysBetween`, `GetDay`, `GetHour`, `GetMinute`, `ToDate`, and others

Pig Latin

Outline

Foreach

`foreach` takes a set of expressions and applies them to every record in the data pipeline, it is the projecting and transforming operation

Loading and transforming

```
A = load 'input' as (user:chararray, id:long, address:chararray, phone:chararray,
                      preferences:map[]);
B = foreach A generate user, id;
```

It is convenient to use expressions in `foreach` statement

Loading and transforming

```
prices = load 'NYSE_daily' as (exchange, symbol, date, open, high, low, close,
                               volume, adj_close);
gain = foreach prices generate close - open;
gain2 = foreach prices generate $6 - $3;
```

Foreach

Some other expressions

Loading and transforming

```
prices = load 'NYSE_daily' as (exchange, symbol, date, open, high, low,
                               close, volume, adj_close);
beginning = foreach prices generate ..open;
-- produces exchange, symbol, date, open

middle = foreach prices generate open..close;
-- produces open, high, low, close

end = foreach prices generate volume..;
-- produces volume, adj_close

all_in_one = foreach prices generate *;
-- produces a tuple of all fields
```

To extract data from complex types

Extracting data from complex types

```
bball = load 'baseball' as (name:chararray, team:chararray,
                            position:bag{t:(p:chararray)}, bat:map[]);
bball_avg = foreach bball generate bat#'batting_average';
bball_p = foreach bball generate position.p;
```

Pig Latin

Outline

Filter

`filter` statement allows you to select which records will be retained in your data pipeline

A filter contains a predicate

If a predicate evaluates to true for a given record, that record will be passed down the pipeline

Otherwise, it will not

Loading and filtering

```
divs = load 'NYSE_dividends' as (exchange:chararray, symbol:chararray,
                                  date:chararray, dividends:float);
startswithcm = filter divs by symbol matches 'CM.*';
-- Only records matching the pattern 'CM.*' are retained
```

Pig Latin

Outline

Group

`group` statement collects together records with the same key

It is sharing its syntax with SQL, but the grouping operator in Pig Latin is fundamentally different than the one in SQL

In SQL `GROUP BY` clause creates a group that must feed directly into one or more aggregate functions

In Pig Latin there is no direct connection between `group` and aggregate functions

Instead, `group` does exactly what it says, i.e. it collects all records with the same value for the provided key together into a bag

Loading and grouping

```
daily = load 'NYSE_daily' as (exchange:chararry, stock_id:int);
grpd = group daily by stock_id;
describe grpd;
grpd: {group: int, daily: {exchange: chararry,stock_id: int}}
```

Group

Group supports multiple keys

The resulting records still have two fields and `group` field is a tuple with a field for each key

```
                                                                Loading and grouping
daily = load 'NYSE_daily' as (exchange, stock, date, dividends);
grpd = group daily by (exchange, stock);
avg = foreach grpd generate group, daily;
describe grpd;
grpd: {group: (exchange: bytearray,stock: bytearray),daily: {exchange: bytearray,
                                                              stock: bytearray,
                                                              date: bytearray,
                                                              dividends: bytearray}}
```

It is also possible to use `all` to group together all of the records in a pipeline

```
                                                                Loading and grouping
daily = load 'NYSE_daily' as (exchange, stock);
grpd = group daily all;
--grpd will have only one row containing all records
```

Pig Latin

Outline

Order By

`order by` statement sorts producing a total order of output data

The syntax of `order by` is similar to `group` statement

Data is sorted based on the types of the indicated fields: numeric values are sorted numerically, chararray fields are sorted lexically

Loading and sorting

```
daily = load 'NYSE_daily' as (exchange:chararray, symbol:chararray, date:chararray,
                              open:float, close:float);
byclose = order daily by close desc, open;

dump byclose;
-- first close in descending order, then open sorted in ascending order
```

Pig Latin

Outline

Distinct

`distinct` statement removes duplicate records

Loading and removing duplicates

```
daily = load 'NYSE_daily' as (exchange:chararray, symbol:chararray);
uniq = distinct daily;
```

Distinct statement operates only on entire records and not on individual fields

Pig Latin

Outline

Joins

`join` operations in Pig are analogous to the join operations in SQL

`join` operations combine records from two bags based on a common field

`join` operations act on two different data sets where one field in each data set is nominated as a join key

In a `join`, the first (second) dataset specified is called as the left (right) entity or data set

Joins

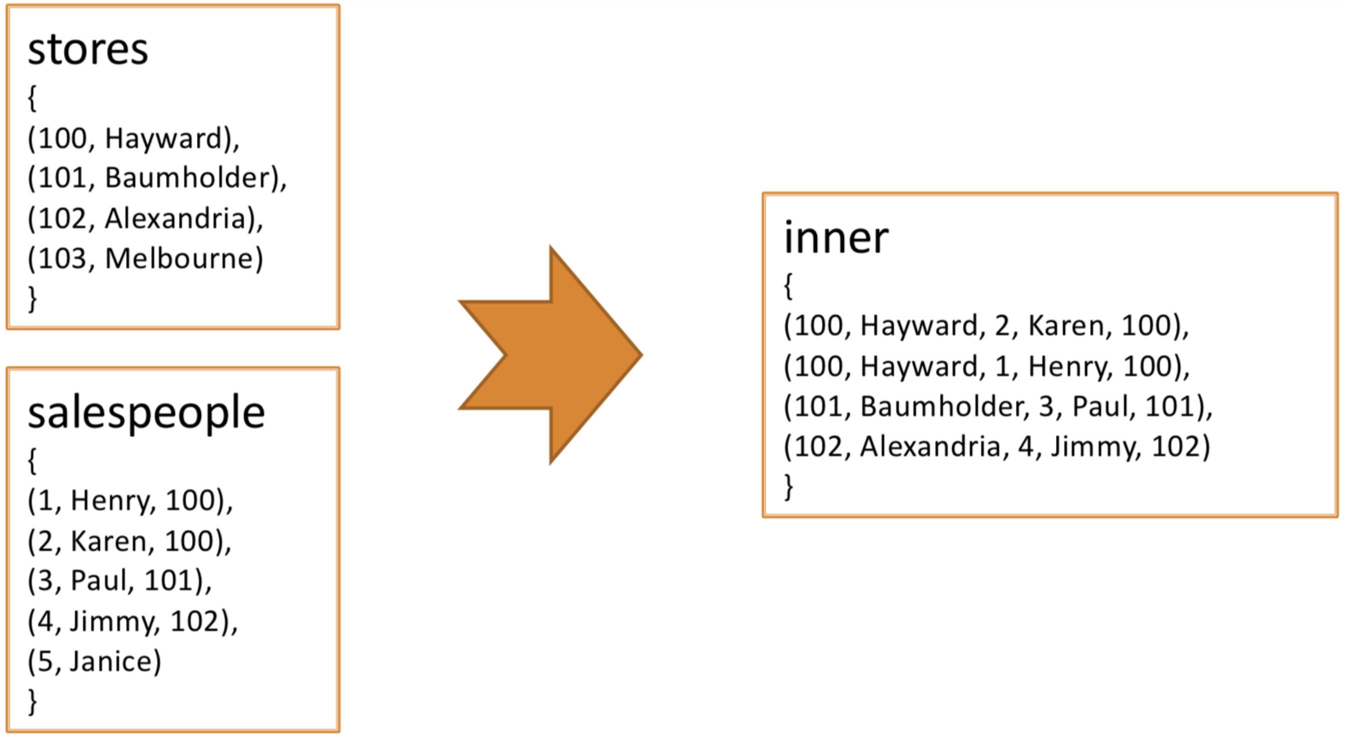
Types of `join` operation

- `inner join`, often simply called a join, returns all elements or records from both datasets where the nominated key is present in both datasets
- `left outer join` returns all records from the left, or first dataset along with matched records only (by the specified key) from the right, or second, dataset
- `right outer join` returns all records from the right, or second dataset along with matched records only (by the specified key) from the left, or first dataset
- `full outer join` returns all records from both datasets whether there is a key match or not

Inner Join

Inner join

```
inner = JOIN stores BY storied, salespeople BY storeid;
```



stores
{
(100, Hayward),
(101, Baumholder),
(102, Alexandria),
(103, Melbourne)
}

salespeople
{
(1, Henry, 100),
(2, Karen, 100),
(3, Paul, 101),
(4, Jimmy, 102),
(5, Janice)
}

inner
{
(100, Hayward, 2, Karen, 100),
(100, Hayward, 1, Henry, 100),
(101, Baumholder, 3, Paul, 101),
(102, Alexandria, 4, Jimmy, 102)
}

Left Outer Join

Left outer join

```
leftouter = JOIN stores BY storied LEFT OUTER, salespeople BY storeid;
```

stores
{
(100, Hayward),
(101, Baumholder),
(102, Alexandria),
(103, Melbourne)
}

salespeople
{
(1, Henry, 100),
(2, Karen, 100),
(3, Paul, 101),
(4, Jimmy, 102),
(5, Janice)
}

leftouter
{
(100, Hayward, 2, Karen, 100),
(100, Hayward, 1, Henry, 100),
(101, Baumholder, 3, Paul, 101),
(102, Alexandria, 4, Jimmy, 102)
(103, Melbourne, , ,)
}

Right Outer Join

Right outer join

```
rightouter = JOIN stores BY storied RIGHT OUTER, salespeople BY storeid;
```

stores
```
{
(100, Hayward),
(101, Baumholder),
(102, Alexandria),
(103, Melbourne)
}
```

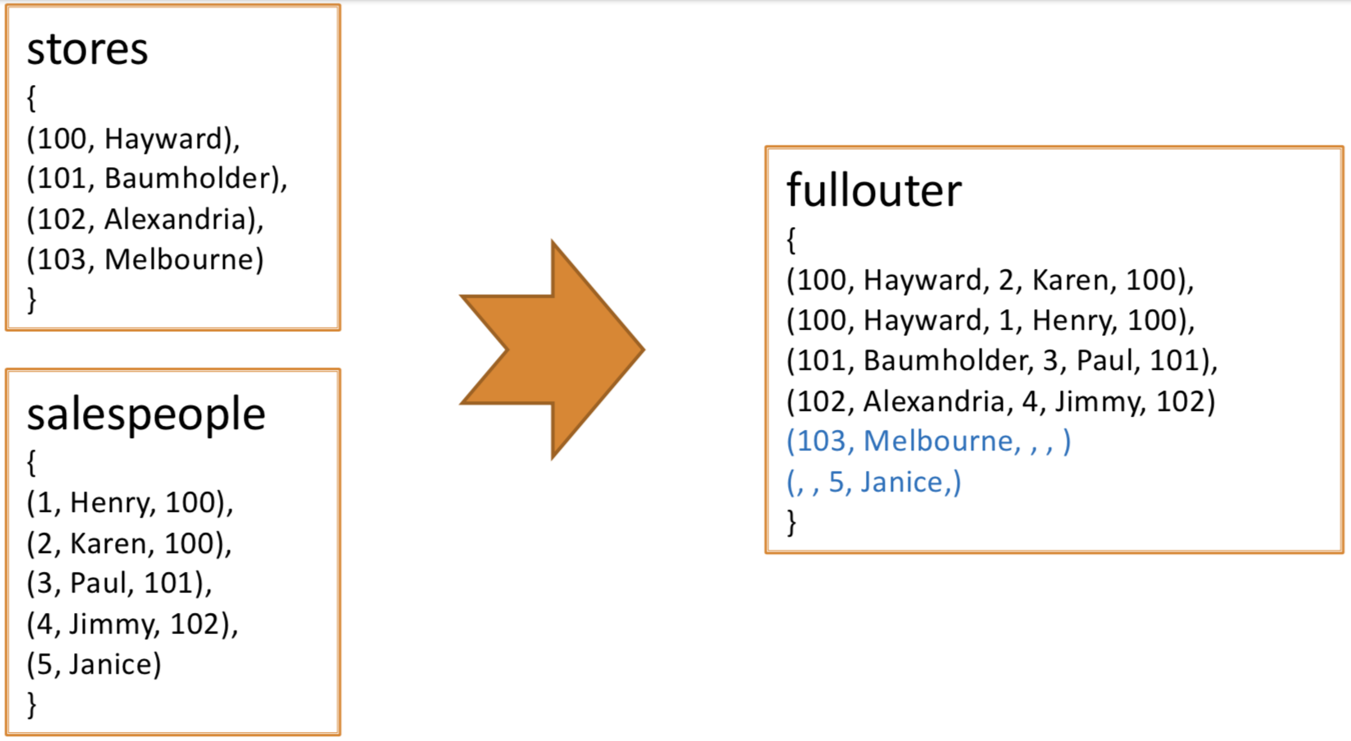
salespeople
```
{
(1, Henry, 100),
(2, Karen, 100),
(3, Paul, 101),
(4, Jimmy, 102),
(5, Janice)
}
```

rightouter
```
{
(100, Hayward, 2, Karen, 100),
(100, Hayward, 1, Henry, 100),
(101, Baumholder, 3, Paul, 101),
(102, Alexandria, 4, Jimmy, 102)
(, , 5, Janice , )
}
```

Full Outer Join

Full outer join

```
fullouter = JOIN stores BY storied FULL OUTER, salespeople BY storeid;
```



```
stores
{
(100, Hayward),
(101, Baumholder),
(102, Alexandria),
(103, Melbourne)
}
```

```
salespeople
{
(1, Henry, 100),
(2, Karen, 100),
(3, Paul, 101),
(4, Jimmy, 102),
(5, Janice)
}
```

```
fullouter
{
(100, Hayward, 2, Karen, 100),
(100, Hayward, 1, Henry, 100),
(101, Baumholder, 3, Paul, 101),
(102, Alexandria, 4, Jimmy, 102)
(103, Melbourne, , , )
(, , 5, Janice,)
}
```

Pig Latin

Outline

Limit and Sample

Sometimes we want to see only a limited number of results

`limit` statement allows for the restriction of an output to a given number of the first records

The following returns at most 10 lines, i.e., the first 10 records

Loading and displaying limited number of records

```
divs = load 'NYSE_dividends';
first10 = limit divs 10;
```

Sometimes we want to see only a limited number of randomly selected results

`sample` statement allows for the restriction of an output to a given percentage of the total number of records

In the following example, 0.1 indicates 10%

Loading and displaying sample number of records

```
divs = load 'NYSE_dividends';
some = sample divs 0.1;
```

References

Gates A., Programming Pig, O'Reilly Media, Inc., 2011, (Available in `READINGS` folder)

Vaddeman B., Beginning Apache Pig: Big Data Processing Made Easy, Apress 2016 (Available in `READINGS` folder)