
BiG
DATA

Class 6 - Pig

Topic 1

Processing data using Apache Pig

AGENDA

- What is Big Data?
- Hadoop Distributed File System
- MapReduce
- Apache Flume
- Apache Sqoop
- **Apache Pig**

Apache Pig

Relational Operators: JOIN

Example:

Customers

customerid	name
1	John
2	Jen
3	Dave
4	Ed
5	Steffi
6	Judith

Orders

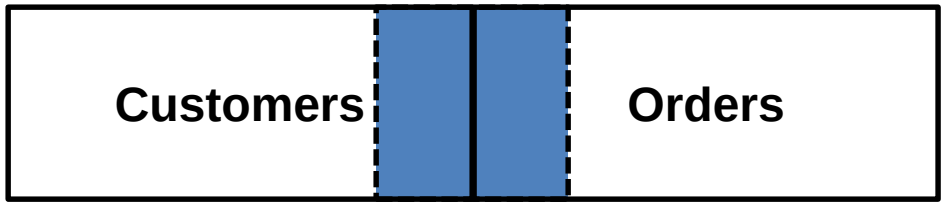
orderid	orderno	customerid
1	1000	1
2	1001	2
3	1002	2
4	1003	1
5	1004	4
6	1005	5

Apache Pig

Relational Operators: JOIN

Example: List out all Customers who have placed an Order.

customers = LOAD '/pig/joins/customers' AS (customerid:int, name:chararray);
orders = LOAD '/pig/joins/orders' AS (orderid:int, orderno:int, customerid:int);



joined = JOIN customers BY customerid, orders BY customerid;

INNER JOIN

Apache Pig

Relational Operators: JOIN

JOIN based on multiple keys:

```
joined = JOIN D1 BY (K1, K2), D2 BY (K1, K2);
```

JOIN multiple datasets:

```
D1 = LOAD 'input1' as (x, y);
D2 = LOAD 'input2' as (u, v);
D3 = LOAD 'input3' as (e, f);

joined = JOIN D1 by x, D2 by u, D3 by e;
```

Apache Pig

How to access fields after a JOIN?

Find list of customers who have placed more than 25 orders:

```
customers = LOAD '/pig/joins/customers' AS (customerid:int,
name:chararray);

orders = LOAD '/pig/joins/orders' AS (orderid:int, orderno:int,
customerid:int);

joined = JOIN customers BY customerid, orders BY customerid;

grouped = GROUP joined BY customers::customerid;

filtered = FILTER grouped BY COUNT(joined) > 25;
```

Apache Pig

Different types of JOINS

OUTER Joins

- Records that do not have a match will also be included.
- Null values are populated for missing fields.

Types:

- LEFT
- RIGHT
- FULL

Apache Pig

Different types of JOINS

LEFT OUTER Join

- Records from the LEFT side will be included even if they do not have a match on the right side.

Syntax:

leftouterjoin = JOIN D1 BY id **LEFT OUTER**, D2 BY id;

Apache Pig

Different types of JOINS

RIGHT OUTER Join

- Records from the RIGHT side will be included even if they do not have a match on the left side.

Syntax:

rightouterjoin = JOIN D1 BY (id1, id2) **RIGHT OUTER**, D2 BY (id1, id2);

Apache Pig

Different types of JOINS

FULL OUTER Join

- Records from BOTH sides will be included even if they do not have matches.

Syntax:

fullouterjoin = JOIN D1 BY id **FULL OUTER**, D2 BY id;

Apache Pig

Different types of JOINS

OUTER Joins

- Pig needs to populate nulls for the data set on the ‘other’ side when there is no match. So the schema for that data set is mandatory.
- LEFT Outer: Schema for **right** side data set mandatory.
- RIGHT Outer: Schema for **left** side data set mandatory.
- FULL Outer: Schema for **both** data sets mandatory.

Apache Pig

COGROUP

- Group two or more datasets by a column and join based on the same column.
- COGROUP on one dataset is same as GROUP.
- COGROUP on multiple datasets results in a record with a key and one bag per dataset.

Example:

cogrouped = **COGROUP** orders **BY** customerid, customers by customerid;

BiG DATA

RECAP

JOINS

COGROUP

Class 6 - Pig

Topic 2

Relational Operators

AGENDA

- What is Big Data?
- Hadoop Distributed File System
- MapReduce
- Apache Flume
- Apache Sqoop
- **Apache Pig**

Apache Pig

Relational Operators: UNION

- Used to concatenate two datasets together.

Syntax:

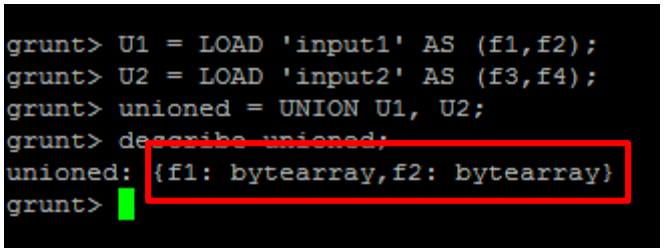
```
U1 = LOAD ‘input1’;
U2 = LOAD ‘input2’;

unioned = UNION U1, U2;
```

Apache Pig

Relational Operators: UNION

Schema is same for both datasets:

U1 = LOAD ‘input1’ AS (f1,f2);
U2 = LOAD ‘input2’ AS (f3,f4);
unioned = UNION U1, U2;
describe unioned;



Result Schema:

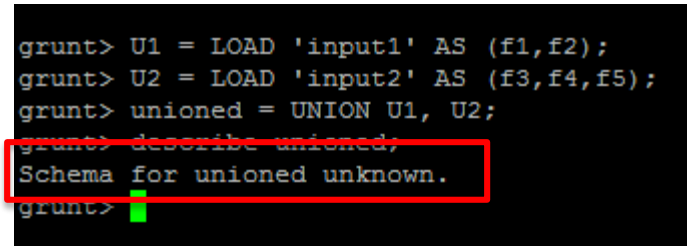
- Same as input
- Names of fields will be that of first dataset

Apache Pig

Relational Operators: UNION

Schema is different:

```
U1 = LOAD 'input1' AS (f1,f2);
U2 = LOAD 'input2' AS (f3,f4,f5);
unioned = UNION U1, U2;
describe unioned;
```



Result Schema:

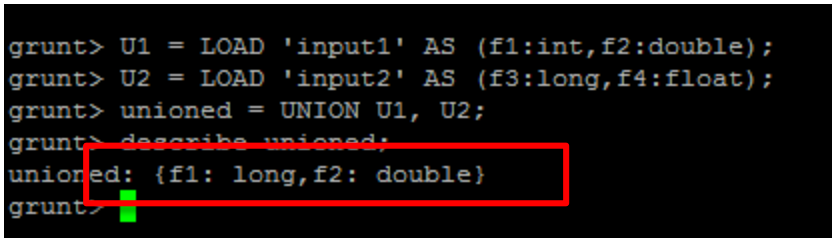
- Schema unknown
- Different records will have different fields

Apache Pig

Relational Operators: UNION

Number of fields are same, but data types are different:

```
U1 = LOAD 'input1' AS (f1:int,f2:double);
U2 = LOAD 'input2' AS (f3:long,f4:float);
unioned = UNION U1, U2;
describe unioned;
```



Apache Pig

Relational Operators: UNION

Number of fields are same, but data types are different:

- Pig escalates data types.

double >> float >> long >> int >> bytearray

tuple >> bag >> map >> chararray >> bytearray

```
grunt> U1 = LOAD 'input1' AS (f1:int,f2:double);
grunt> U2 = LOAD 'input2' AS (f3:long,f4:float);
grunt> unioned = UNION U1, U2;
grunt> describe unioned;
unioned: {f1: long,f2: double}
grunt>
```

Apache Pig

Relational Operators: UNION

Data types are different and incompatible:

```
U1 = LOAD 'input1' AS (f1:int,f2:double);
U2 = LOAD 'input2' AS (f3:chararray,f4:float);
unioned = UNION U1, U2;
describe unioned;
```

```
grunt> U1 = LOAD 'input1' AS (f1:int,f2:double);
grunt> U2 = LOAD 'input2' AS (f3:chararray,f4:float);
grunt> unioned = UNION U1, U2;
2014-08-06 13:57:59,739 [main] ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1051: Cannot cast to bytearray
Details at logfile: /home/hduser/pig_1407346417739.log
grunt>
```

Result Schema:

- Error: Cannot cast to bytearray

Apache Pig

Relational Operators: UNION

How to perform UNION on incompatible data types:

```
U1 = LOAD 'input1' AS (f1:int,f2:double);
U1a = FOREACH U1 GENERATE (chararray) f1, f2;
U2 = LOAD 'input2' AS (f3:chararray,f4:float);

unioned = UNION U1a, U2;
describe unioned;
```

Apache Pig

Relational Operators: UNION

How to perform UNION on incompatible data types:

U1 = LOAD ‘input1’ AS (f1:int,f2:double);
U2 = LOAD ‘input2’ AS (f2:float,f3:float);
unioned = UNION ONSCHEMA U1, U2;
describe unioned;

```
grunt> U1 = LOAD 'input1' AS (f1:int,f2:double);
grunt> U2 = LOAD 'input2' AS (f2:float,f3:float);
grunt> unioned = UNION ONSCHEMA U1, U2;
grunt> describe unioned;
unioned: {f1: int,f2: double,f3: float}
grunt>
```

ONSCHEMA:

- Matching of field by name.
- Field does not match, then it is added to output.
- All inputs must have schemas.

Apache Pig

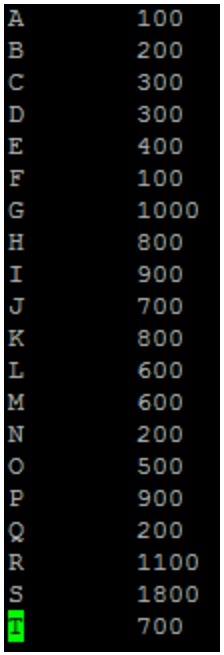
Relational Operators: RANK

- Used to rank each tuple in a relation.

Example:

```
sales_data = LOAD 'rank' AS (name, sales);
ranked = RANK sales_data;
dump ranked;
```

Input



Apache Pig

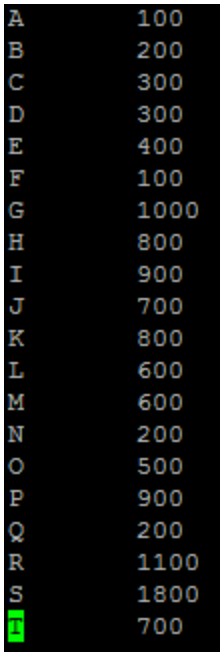
Relational Operators: RANK

- Used to rank each tuple in a relation.

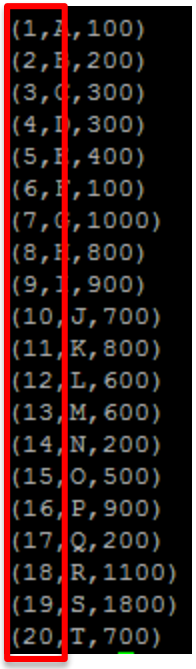
Example:

```
sales_data = LOAD 'rank' AS (name, sales);
ranked = RANK sales_data;
dump ranked;
```

Input



Output



Apache Pig

Relational Operators: RANK

Rank based on field:

```
sales_data = LOAD 'rank' AS (name, sales);
ranked = RANK sales_data BY sales DESC;
dump ranked;
```

Input

```
A       100
B       200
C       300
D       300
E       400
F       100
G       1000
H       800
I       900
J       700
K       800
L       600
M       600
N       200
O       500
P       900
Q       200
R       1100
S       1800
T       700
```

Output

```
(1,R,100)
(1,A,100)
(3,C,1000)
(4,R,1100)
(5,S,1800)
(6,Q,200)
(6,N,200)
(6,F,200)
(9,C,300)
(9,D,300)
(11,E,400)
(12,O,500)
(13,L,600)
(13,M,600)
(15,T,700)
(15,J,700)
(17,H,800)
(17,K,800)
(19,I,900)
(19,P,900)
```

Apache Pig

Relational Operators: RANK

Rank based on field and avoid gaps in ranks:

```
sales_data = LOAD 'rank' AS (name, sales);
ranked = RANK sales_data BY sales DESC DENSE;
dump ranked;
```

Input

```
A	100
B	200
C	300
D	300
E	400
F	100
G	1000
H	800
I	900
J	700
K	800
L	600
M	600
N	200
O	500
P	900
Q	200
R	1100
S	1800
T	700
```

Output

```
(1,F,100)
(1,A,100)
(3,C,1000)
(4,R,1100)
(5,S,1800)
(6,Q,200)
(6,N,200)
(6,B,200)
(9,C,300)
(9,D,300)
(11,E,400)
(12,O,500)
(13,L,600)
(13,M,600)
(15,T,700)
(15,J,700)
(17,H,800)
(17,K,800)
(19,I,900)
(19,P,900)
```

Class 6 - Pig

Topic 3

Parameter Substitution

AGENDA

- What is Big Data?
- Hadoop Distributed File System
- MapReduce
- Apache Flume
- Apache Sqoop
- **Apache Pig**

Apache Pig

Parameter Substitution

- Pig Latin scripts will have data elements that change dynamically, like a Date field.
- Scripts should not be modified when such field values change.
- Pig provides option to pass values as parameters.
- Uses string replacement functionality.

Methods:

- Pass parameters on command line.
- Pass parameters from a file.

Apache Pig

Parameter Substitution: Command Line

Find stock prices by date:

```
input_daily = load '/pig/NYSE_daily' as (exchange:chararray,
symbol:chararray, date:chararray, open:float, high:float, low:float,close:float,
volume:int, adj_close:float);
stock_prices = filter input_daily by date == '$date';
```

- Save scripts as paramtest.pig

Command:

pig -p date=3/17/2009 **paramtest.pig**

Multiple Parameters:

pig -p <param1>=<value1> -p <param2>=<value2> scripts.pig

Apache Pig

Parameter Substitution: parameters file

- Useful when more than a few parameters need to be configured.
- One parameter per line:

```
parameter1 = value1
parameter2 = value2
parameter3 = value3
```

Apache Pig

Parameter Substitution: parameters file

Find stock prices by date and greater than a certain threshold value:

```
input_daily = load '/pig/NYSE_daily' as (exchange:chararray, symbol:chararray,
date:chararray, open:float, high:float, low:float, close:float, volume:int,
adj_close:float);
stock_prices = filter input_daily by date == '$date' AND close > $threshold;
dump stock_prices;
```

Sample parameters file:

```
date=3/17/2009
threshold=5
```

Command to run:

```
pig -param_file params-file paramtest-file.pig
```

Apache Pig

Parameter Substitution: parameters file

Find stock prices by date and greater than a certain threshold value:

```
input_daily = load '/pig/NYSE_daily' as (exchange:chararray, symbol:chararray,
date:chararray, open:float, high:float, low:float, close:float, volume:int,
adj_close:float);
stock_prices = filter input_daily by date == '$date' AND close > $threshold;
dump stock_prices;
```

Sample parameters file:

```
date=3/17/2009
threshold=5
```

Command to run:

```
pig -param_file params-file paramtest-file.pig
```

RECAP

Parameter Substitution

RECAP

UNION
RANK

Class 6 - Pig

Topic 4

User Defined Functions

AGENDA

- What is Big Data?
- Hadoop Distributed File System
- MapReduce
- Apache Flume
- Apache Sqoop
- **Apache Pig**

Apache Pig

User Defined Functions

- Built-in Functions
- Custom User Defined Functions
  - Functions that come with Pig
  - Functions written by other users
  - Writing your own functions

Apache Pig

Built-In Functions

- Eval Functions
- Math Functions
- String Functions
- Date Functions
- Tuple Functions
- Load/Store Functions

Apache Pig

Built-In Functions: AVG

- Used to compute the average of the numeric values in a single-column bag.
- Requires a preceding GROUP statement to compute averages.

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

grouped = GROUP input_divs BY symbol;

average = FOREACH grouped GENERATE group,
AVG(input_divs.dividend);
```

Apache Pig

Built-In Functions: COUNT

- Used to count the number of elements in a bag.
- Requires a preceding GROUP statement to find counts.
- If first field of tuple is NULL, it will NOT be counted.

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

grouped = GROUP input_divs BY symbol;

counted = FOREACH grouped GENERATE group, COUNT(input_divs);
```

Apache Pig

Built-In Functions: COUNT_STAR

- Similar to COUNT function.
- Counts even the NULL values.

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

grouped = GROUP input_divs BY symbol;

counted = FOREACH grouped GENERATE group,
COUNT_STAR(input_divs);
```

Apache Pig

Built-In Functions: CONCAT

- Concatenate two fields or expressions.
- Data types of the fields or expressions should be the same.

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

concatenated = FOREACH input_divs GENERATE CONCAT(exchange,
symbol);

concatenated = FOREACH input_divs GENERATE
CONCAT(CONCAT(exchange, '-'), symbol);
```

Apache Pig

Built-In Functions: DIFF

- Used to compare two fields in a tuple.
- Tuples in one bag but not in the other are returned in a bag.

Syntax:

```
input = LOAD 'input' AS (B1:bag{T1:tuple(t1:int,t2:int)},
B2:bag{T2:tuple(f1:int,f2:int)});

bagged = FOREACH input DIFF(B1,B2);
```

Apache Pig

Built-In Functions: IsEmpty

- Used to check if a bag or map is empty.
- Filter empty data.

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

nonempty = FILTER input_divs BY NOT IsEmpty(date);
```

Apache Pig

Built-In Functions: MAX/MIN

- Used to compute maximum/minimum of numeric or chararray values in a bag.
- Requires a preceding GROUP statement.

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

grouped = GROUP input_divs BY symbol;

maximum = FOREACH grouped GENERATE input_divs.symbol,
MAX(input_divs.dividend);

minimum = FOREACH grouped GENERATE input_divs.symbol,
MIN(input_divs.dividend);
```

Apache Pig

Built-In Functions: SIZE

- Used to compute the number of elements of any Pig Data type.
- Includes NULL values.

Data Type	Return value
int long float double	Value 1
chararray	Number of characters
bytearray	Number of bytes
tuple	Number of fields
bag	Number of tuples
map	Number of key/value pairs

Apache Pig

Built-In Functions: SIZE

Example:

```
input_divs = LOAD '/pig/NYSE_dividends' AS (exchange:chararray,
symbol:chararray, date:chararray, dividend:float);

size = FOREACH input_divs GENERATE SIZE(exchange), SIZE(dividend);
```

Apache Pig

User Defined Functions

- Functions that come with Pig
- Functions written by other users
- Writing your own functions

Apache Pig

Piggy Bank

- Pig's repository of user-contributed functions
- Piggybank functions are distributed as part of Pig distribution, but not built-in.
- Functions contributed as-is
- Bugs or missing functionality should be added by you

Shared code path:

http://svn.apache.org/viewvc/pig/trunk/contrib/piggybank/java/src/main/java/org/apache/pig/piggybank/

Apache Pig

Registering UDFs

- Reverse UDF

```
register 'pig/contrib/piggybank/java/piggybank.jar';

input = LOAD ‘/pig/reverse.txt’;

reversed = FOREACH input GENERATE
org.apache.pig.piggybank.evaluation.string.Reverse($0);
```

Apache Pig

Registering UDFs: Define

```
register 'pig/contrib/piggybank/java/piggybank.jar';

define reverse org.apache.pig.piggybank.evaluation.string.Reverse();

input = LOAD ‘/pig/reverse.txt’;

reversed = FOREACH input GENERATE reverse($0);
```

Apache Pig

Registering UDFs: Command Line

pig -Dudf.import.list=org.apache.pig.piggybank.evaluation.string
register_script.pig

register_script.pig:

```
register 'pig/contrib/piggybank/java/piggybank.jar';

input = LOAD ‘/pig/reverse.txt’;

reversed = FOREACH input GENERATE Reverse($0);
```

Apache Pig

User Defined Functions: Apache Data Fu

Apache DataFu: Collection of libraries for working with large-scale data in Hadoop.

https://github.com/linkedin/datafu

User Defined Functions for:

- Statistics
- Bag Operations
- Sampling
- Estimation
- Hashing

Apache Pig

Write your own UDF

Eval: Typically used to alter a field in the data pipeline such as parsing a string.

Filter: Used to filter a record in the data pipeline.

Group/Aggregate: Used to build aggregation functions.

Load/Store: Used to control how data is loaded into or stored out of Pig.

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Write your own UDF: Factorial

```
public class Factorial extends EvalFunc<Long> {

        public Long exec(Tuple input) throws IOException {
                try {
                                int fact = (Integer)input.get(0);
                                long result = 1;

                                for (int i = 1; i <= fact; i++) {
                                result = result * i;
                                }
                                return result;
                        }
                catch (Exception e) {
        //Throwing an exception will cause the task to fail.
                                throw new IOException("Something bad happened!", e);
                }
        }
}
```

Apache Pig

Calling your UDF

```
register /home/hduser/jars/PigUDFs.jar

fact = LOAD '/pig/factorial' AS (val:int);

result = FOREACH fact GENERATE com.jigsaw.udfs.Factorial(val);

dump result;
```

RECAP

User Defined Functions
How to write a simple UDF?