Operators

Basic Operators

Operator	Description	Example
Arithmetic Operators	+, -, *, /, %, ?:	X = FOREACH A GENERATE f1, f2, f1%f2; X = FOREACH A GENERATE f2, (f2==1?1:COUNT(B));
Boolean Operators	and, or, not	X = FILTER A BY (f1==8) OR (NOT (f2+f3 > f1));
Cast Operators	Casting from one datatype to another	B = FOREACH A GENERATE (int)$0 + 1; B = FOREACH A GENERATE $0 + 1, $1 + 1.0
Comparison Operators	==, !=, >, <, >=, <=, matches	X = FILTER A BY (f1 == 8); X = FILTER A BY (f2 == 'apache'); X = FILTER A BY (f1 matches '.*apache.*');
Construction Operators	Used to construct tuple (), bag {} and map []	B = foreach A generate (name, age); B = foreach A generate {(name, age)}, {name, age}; B = foreach A generate [name, gpa];
Dereference Operators	dereference tuples (tuple.id or tuple.(id,…)), bags (bag.id or bag.(id,…)) and maps (map#'key')	X = FOREACH A GENERATE f2.t1,f2.t3 (dereferencing is used to retrieve two fields from tuple f2)
Disambiguate Operator	(::) used to identify field names after JOIN, COGROUP, CROSS, or FLATTEN operators	A = load 'data1' as (x, y); B = load 'data2' as (x, y, z); C = join A by x, B by x; D = foreach C generate A::y;
Flatten Operator	Flatten un-nests tuples as well as bags	consider a relation that has a tuple of the form (a, (b, c)). The expression GENERATE $0, flatten($1), will cause that tuple to become (a, b, c).
Null Operator	is null, is not null	X = FILTER A BY f1 is not null;
Sign Operators	+ -> has no effect, - -> changes the sign of a positive/negative number	A = LOAD 'data' as (x, y, z); B = FOREACH A GENERATE -x, y;

Relational Operators

Operator	Description	Example
COGROUP/GROUP	Groups the data in one or more relations. The COGROUP operator groups together tuples that have the same group key (key field)	A = load 'student' AS (name:chararray,age:int,gpa:float); B = GROUP A BY age;
CROSS	Computes the cross product of two or more relations	X = CROSS A,B DUMP X; (1,2,3,2,4) (1,2,3,8,9) (1,2,3,1,3) (4,2,1,2,4) (4,2,1,8,9) (4,2,1,1,3) A = (1, 2, 3) (4, 2, 1) B = (2, 4) (8, 9) (1, 3)
DEFINE	Assigns an alias to a UDF or streaming command.	DEFINE CMD `perl PigStreaming.pl - nameMap` input(stdin using PigStreaming (',')) output (stdout using PigStreaming(',')); A = LOAD 'file'; B = STREAM B THROUGH CMD;
DISTINCT	Removes duplicate tuples in a relation.	X = DISTINCT A; DUMP X; (1,2,3) (4,3,3) (8,3,4) A = (8,3,4) (1,2,3) (4,3,3) (4,3,3) (1,2,3)
FILTER	Generates transformation of data for each row as specified	X = FILTER A BY f3 == 3; DUMP X; (1,2,3) (4,3,3) (8,4,3) A = (1,2,3) (4,5,6) (7,8,9) (4,3,3) (8,4,3)
FOREACH	Selects tuples from a relation based on some condition.	X = FOREACH A GENERATE a1, a2; DUMP X; (1,2) (4,2) (8,3) A = (1,2,3) (4,2,5) (8,3,6)
IMPORT	Import macros defined in a separate file.	/* myscript.pig */ IMPORT 'my_macro.pig';

Relational Operators

Operator	Description	Example
JOIN	Performs an inner join of two or more relations based on common field values.	X = JOIN A BY a1, B BY b1; DUMP X (1,2,1,3) A = (1,2) B = (1,3) (1,2,1,2) (4,5) (1,2) (4,5,4,7) (4,7)
LOAD	Loads data from the file system.	A = LOAD 'myfile.txt'; LOAD 'myfile.txt' AS (f1:int, f2:int, f3:int);
MAPREDUCE	Executes native MapReduce jobs inside a Pig script.	A = LOAD 'WordcountInput.txt'; B = MAPREDUCE 'wordcount.jar' STORE A INTO 'inputDir' LOAD 'outputDir' AS (word:chararray, count: int) `org.myorg.WordCount inputDir outputDir`;
ORDERBY	Sorts a relation based on one or more fields.	A = LOAD 'mydata' AS (x: int, y: map[]); B = ORDER A BY x;
SAMPLE	Partitions a relation into two or more relations, selects a random data sample with the stated sample size.	Relation X will contain 1% of the data in relation A. A = LOAD 'data' AS (f1:int,f2:int,f3:int); X = SAMPLE A 0.01;
SPLIT	Partitions a relation into two or more relations based on some expression.	SPLIT input_var INTO output_var IF (field1 is not null), ignored_var IF (field1 is null);
STORE	Stores or saves results to the file system.	STORE A INTO 'myoutput' USING PigStorage ('*'); 1*2*3 4*2*1
STREAM	Sends data to an external script or program	STORE A INTO 'myoutput' USING PigStorage ('*'); 1*2*3 4*2*1
UNION	Computes the union of two or more relations. (Does not preserve the order of tuples)	X = UNION A, B; A = (1,2,3) B = (2,4) DUMP X; (4,2,1) (8,9) (1,2,3) (1,3) (4,2,1) (2,4) (8,9) (1,3)

Functions

Eval Functions

Function	Syntax	Description
AVG	AVG(expression)	Computes the average of the numeric values in a single-column bag.
CONCAT	CONCAT (expression, expression)	Concatenates two expressions of identical type.
COUNT	COUNT(expression)	Computes the number of elements in a bag, it ignores null.
COUNT_STAR	COUNT_STAR(expression)	Computes the number of elements in a bag, it includes null.
DIFF	DIFF (expression, expression)	Compares two fields in a tuple, any tuples that are in one bag but not the other are returned in a bag.
IsEmpty	IsEmpty(expression)	Checks if a bag or map is empty.
MAX	MAX(expression)	Computes the maximum of the numeric values or chararrays in a single-column bag
MIN	MIN(expression)	Computes the minimum of the numeric values or chararrays in a single-column bag.
SIZE	SIZE(expression)	Computes the number of elements based on any Pig data type. SIZE includes NULL values in the size computation
SUM	SUM(expression)	Computes the sum of the numeric values in a single-column bag.
TOKENIZE	TOKENIZE(expression [, 'field_delimiter'])	Splits a string and outputs a bag of words.

Load/Store Functions

Function	Syntax	Description
Handling Compression	A = load 'myinput.gz'; store A into 'myoutput.gz';	PigStorage and TextLoader support gzip and bzip compression for both read (load) and write (store). BinStorage does not support compression.

Load/Store Functions

Function	Syntax	Description
BinStorage	A = LOAD 'data' USING BinStorage();	Loads and stores data in machine-readable format.
JsonLoader, JsonStorage	A = load 'a.json' using JsonLoader();	Load or store JSON data.
PigDump	STORE X INTO 'output' USING PigDump();	Stores data in UTF-8 format.
PigStorage	A = LOAD 'student' USING PigStorage('\t') AS (name: chararray, age:int, gpa: float);	Loads and stores data as structured text files.
TextLoader	A = LOAD 'data' USING TextLoader();	Loads unstructured data in UTF-8 format.

Math Functions

Function	Syntax	Description
ABS	ABS(expression)	Returns the absolute value of an expression. If the result is not negative ($x \geq 0$), the result is returned. If the result is negative ($x < 0$), the negation of the result is returned.
ACOS	ACOS(expression)	Returns the arc cosine of an expression.
ASIN	ASIN(expression)	Returns the arc sine of an expression.
ATAN	ATAN(expression)	Returns the arc tangent of an expression.
CBRT	CBRT(expression)	Returns the cube root of an expression.
CEIL	CEIL(expression)	Returns the value of an expression rounded up to the nearest integer. This function never decreases the result value.
COS	COS(expression)	Returns the trigonometric cosine of an expression.
COSH	COSH(expression)	Returns the hyperbolic cosine of an expression.

Math Functions

Function	Syntax	Description
EXP	EXP(expression)	Returns Euler's number e raised to the power of x.
FLOOR	FLOOR(expression)	Returns the value of an expression rounded down to the nearest integer. This function never increases the result value.
LOG	LOG(expression)	Returns the natural logarithm (base e) of an expression.
LOG10	LOG10(expression)	Returns the base 10 logarithm of an expression.
RANDOM	RANDOM( )	Returns a pseudo random number (type double) greater than or equal to 0.0 and less than 1.0.
ROUND	ROUND(expression)	Returns the value of an expression rounded to an integer (if the result type is float) or rounded to a long (if the result type is double).
SIN	SIN(expression)	Returns the sine of an expression.
SINH	SINH(expression)	Returns the hyperbolic sine of an expression.
SQRT	SQRT(expression)	Returns the positive square root of an expression.
TAN	TAN(expression)	Returns the trignometric tangent of an angle.
TANH	TANH(expression)	Returns the hyperbolic tangent of an expression.

String Functions

Function	Syntax	Description
INDEXOF	INDEXOF(string, 'character', startIndex)	Returns the index of the first occurrence of a character in a string, searching forward from a start index.
LAST_INDEX	LAST_INDEX_OF (expression)	Returns the index of the last occurrence of a character in a string, searching backward from a start index.
LCFIRST	LCFIRST(expression)	Converts the first character in a string to lower case.

String Functions

Function	Syntax	Description
LOWER	LOWER(expression)	Converts all characters in a string to lower case.
REGEX_EXTRACT	REGEX_EXTRACT (string, regex, index)	Performs regular expression matching and extracts the matched group defined by an index parameter. The function uses Java regular expression form.
REGEX_EXTRACT _ALL	REGEX_EXTRACT (string, regex)	Performs regular expression matching and extracts all matched groups. The function uses Java regular expression form.
REPLACE	REPLACE(string, 'oldChar', 'newChar');	Replaces existing characters in a string with new characters.
STRSPLIT	STRSPLIT(string, regex,limit)	Splits a string around matches of a given regular expression.
SUBSTRING	SUBSTRING(string, startIndex, stopIndex)	Returns a substring from a given string.
TRIM	TRIM(expression)	Returns a copy of a string with leading and trailing white space removed.
UCFIRST	UCFIRST(expression)	Returns a string with the first character converted to upper case.
UPPER	UPPER(expression)	Returns a string converted to upper case.

Tuple, Bag, Map Functions

Function	Syntax	Description
TOTUPLE	TOTUPLE(expression [, expression ...])	Converts one or more expressions to type tuple.
TOBAG	TOBAG(expression [, expression ...])	Converts one or more expressions to individual tuples which are then placed in a bag.
TOMAP	TOMAP(key-expression, value-expression[, key-expression, value-expression ...])	Converts key/value expression pairs into a map. Needs an even number of expressions as parameters. The elements must comply with map type rules.
TOP	TOP(topN,column,relation)	Returns the top-n tuples from a bag of tuples.

User Defined Functions (UDFs)

Pig provides extensive support for user defined functions (UDFs) as a way to specify custom processing. Pig UDFs can currently be implemented in three languages: Java, Python, JavaScript and Ruby.

Registering UDFs

Registering Java UDFs

```
---register_java_udf.pig
register 'your_path_to_piggybank/piggybank.jar';
divs     = load 'NYSE_dividends' as (exchange:chararray, symbol:chararray,
date:chararray, dividends:float);
```

Registering Python UDFs (The Python script must be in your current directory)

```
--register_python_udf.pig
register 'production.py' using jython as bballudfs;
players  = load 'baseball' as (name:chararray, team:chararray,
pos:bag{t:(p:chararray)}, bat:map[]);
```

Writing UDFs

Java UDFs

```
packagemyudfs;
importjava.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;

public class UPPER extends EvalFunc<String>
{
  public String exec(Tuple input) throws IOException {
     if (input == null || input.size() == 0)
        return null;
        try{
          String str = (String)input.get(0);
          return str.toUpperCase();
        }catch(Exception e){
          throw new IOException("Caught exception processing input row ", e);
        }
    }
 }
```

Python UDFs

```
#/usr/bin/python

#Square - Square of a number of any data type
@outputSchemaFunction("squareSchema") --Defines a script delegate function that defines schema for
this function depending upon the input type.
def square(num):
  return ((num)*(num))
@schemaFunction("squareSchema") --Defines delegate function and is not registered to Pig.
defsquareSchema(input):
  return input
```

```
#Percent- Percentage
@outputSchema("percent:double") --Defines schema for a script UDF in a format that Pig understands and is able to parse
def percent(num, total):
  return num * 100 / total
```

Data Types

Simple Types

Data Type	Description	Examples
int	Signed 32-bit integer	10
long	Signed 64-bit integer	Data: 10L or 10l Display: 10L
float	32-bit floating point	Data: 10.5F or 10.5f or 10.5e2f or 10.5E2F Display: 10.5F or 1050.0F
double	64-bit floating point	Data: 10.5 or 10.5e2 or 10.5E2 Display: 10.5 or 1050.0
chararray	Character array (string) in Unicode UTF-8 format	hello world
bytearray	Byte array (blob)	
boolean	boolean	true/false (case insensitive)

Complex TypesTypes

Type	Description	Examples
tuple	An ordered set of fields.	(19,2)
bag	An collection of tuples.	{(19,2), (18,1)}
map	A set of key value pairs.	[name#John,phone#5551212]