MORTAR

PIG Cheat Sheet

Contents

We love Apache Pig for data processing—it's easy to learn, it works with all kinds of data, and it plays well with Python, Java, and other popular languages. And, of course, Pig runs on Hadoop, so it's built for high-scale data science.

Whether you're just getting started with Pig or you've already written a variety of Pig scripts, this compact reference gathers in one place many of the tools you'll need to make the most of your data using Pig 0.12.

Additional Resources

Official Pig website
pig.apache.org

***Programming Pig*, by Alan Gates**
chimera.labs.oreilly.com/books/1234000001811

Mortar's Pig page
www.mortardata.com/pig-resources

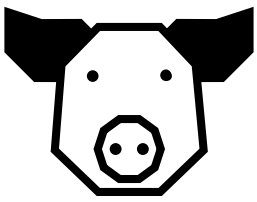
Data Types

Pig is written in Java, so Pig data types correspond to underlying Java data types. When using a UDF (see page 5), the Java types need to be translated into the target language.

	TYPES	DESCRIPTION	EXAMPLE	PYTHON/JYTHON UDF TYPE
SIMPLE	int	Signed 32-bit integer	10	int
	long	Signed 64-bit integer	10L	long
	float	32-bit floating point	10.5f	float
	double	64-bit floating point	10.5	float
	chararray	Character array (string) in Unicode UTF-8 format	hello world	str or unicode
	bytearray	Byte array (blob)		bytearray
	boolean	boolean	true/false	bool
	datetime	org.joda.time.DateTime	1970-01-01T00:00:00.000+00:00	datetime
	biginteger	Java BigInteger	100000000000	long
	bigdecimal	Java BigDecimal	42.4242424242424	float
COMPLEX	tuple	An ordered set of fields.	(17, random)	tuple
	bag	A collection of tuples.	{(17,random), (42,life)}	list
	map	A set of key value pairs.	[foo#bar,baz#quux]	dict

Diagnostic Operators

DESCRIBE	Returns the schema of a relation.
DUMP	Dumps or displays results to screen.
EXPLAIN	Displays execution plans.
ILLUSTRATE	Displays a step-by-step execution of a sequence of statements.



Relational Operators

These operators are the heart of Pig set operations. The fundamental ways to manipulate data appear here, including GROUP, FILTER, and JOIN.

NAME	DESCRIPTION
`COGROUP alias BY (col1, col2)`	COGROUP is the same as GROUP. For readability, programmers usually use GROUP when only one relation is involved and COGROUP when multiple relations are involved.
`CROSS alias1, alias2`	Computes the cross product of two or more relations.
`CUBE alias BY CUBE(exp1, exp2, ...)`	Cube operation computes aggregates for all possbile combinations of specified group by dimensions. The number of group by combinations generated by cube for n dimensions will be 2^n.
`CUBE alias BY ROLLUP(exp1, exp2, ...)`	Rollup operation computes multiple levels of aggregates based on hierarchical ordering of specified group by dimensions. Rollup is useful when there is hierarchical ordering on the dimensions. The number of group by combinations generated by rollup for n dimensions will be n+1.
`DISTINCT alias`	Removes duplicate tuples in a relation.
`FILTER alias BY expression`	Selects tuples from a relation based on some condition.
`...FLATTEN(tuple)` `...FLATTEN(bag)`	Un-nests a bag or a tuple.
`FOREACH ... GENERATE`	Performs transformations on each row in the data.

`GROUP alias ALL`
`GROUP alias BY expression`
`GROUP alias BY (exp1, exp2, ...)`
	Groups the data in one or multiple relations.

`JOIN smaller_alias BY expression, larger_alias BY expression`	Performs inner, equijoin of two or more relations based on common field values.
`JOIN smaller_alias BY expression [LEFT|RIGHT|OUTER], larger_alias BY expression`	Performs an outer join of two or more relations based on common field values.
`JOIN big_alias BY expression, small_alias BY expression USING 'replicated'`	Efficient join when one or more relations are small enough to fit in main memory.
`LIMIT alias n`	Limits the number of output tuples.
`LOAD 'data' [USING function] [AS schema]`	Loads data from the file system.

Relational Operators (cont...)

NAME	DESCRIPTION
`ORDER alias BY col [ASC|DESC]`	Sorts a relation based on one or more fields.
`RANK alias BY col [ASC|DESC]`	Returns each tuple with the rank within a relation.
`SAMPLE alias size`	Selects a random sample of data based on the specified sample size.
`SPLIT alias INTO alias1 IF expression, alias2 IF expression...`	Partitions a relation into two or more relations.
`STORE alias INTO 'directory' [USING function]`	Stores or saves results to the file system.
`UNION alias1, alias2`	Computes the union of two or more relations.

Syntax Tips

The fields in a Pig relation have an explicit order (unlike SQL columns). As a result, there are syntax shortcuts that rely on that field order.

Assume:

```
my_data = LOAD 'file.tsv' AS (field1:int, field2:chararray, field3:float, field4:int);
```

STATEMENT	OUTPUT
`FOREACH my_data GENERATE $0, $2;`	(field1, field3)
`FOREACH my_data GENERATE field2..field4;`	(field2, field3, field4)
`FOREACH my_data GENERATE field2..;`	(field2, field3, field4)
`FOREACH my_data GENERATE *;`	(field1, field2, field3, field4)

How to Use UDFs

Much of Pig's power comes from the fact that it can be extended using other languages. User-defined functions (UDFs) can be written in Java, Python, Jython, Ruby, and javascript.

Python

PIG

```
REGISTER 'udfs.py' USING streaming_python AS udfs;
```

PYTHON

```
@outputSchema("user_products:bag{t:(product_id:long)}")
def deserialize_user_products(product_ids):
    return [ (product_id, ) for product_id in product_ids.split(',') ]
```

Jython

PIG

```
REGISTER 'udfs.py' USING jython AS udfs;
```

JYTHON

```
@outputSchema("user_products:bag{t:(product_id:long)}")
def deserialize_user_products(product_ids):
    return [ (product_id, ) for product_id in product_ids.split(',') ]
```

Java

PIG

```
REGISTER 'udf-project.jar';
DEFINE My_Function my.function.path.MyFunction();
```

JAVA

For an introduction to java UDFs and Loaders, see www.mortardata.com/java-pig

Download the Mortar Pig Java Template for a template project with working pom.xml files.

Ruby

PIG

```
REGISTER 'test.rb' USING jruby AS myfuncs;
```

RUBY

```
require 'pigudf'
 class Myudfs < PigUdf
     outputSchema num:int
     def square num
         return nil if num.nil?
         num**2
     end
 end
```

Mathematical Functions

These functions are very similar to what you can find in java.lang.Math—basic mathematical functions and the trigonometric repertoire.

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
ABS(int a), ABS(long a), ABS(float a), ABS(double a)	int, long, float, double	Returns the absolute value of an expression.
ACOS(double a)	double	Returns the arc cosine of an expression.
ASIN(double a)	double	Returns the arc sine of an expression.
ATAN(double a)	double	Returns the arc tangent of an expression.
CBRT(double a)	double	Returns the cube root of an expression.
CEIL(double a)	double	Returns the value of an expression rounded up to the nearest integer.
COS(double a)	double	Returns the cosine of an expression.
COSH(double a)	double	Returns the hyperbolic cosine of an expression.
EXP(double a)	double	Returns Euler's number e raised to the power of x.
FLOOR(double a)	double	Returns the value of an expression rounded down to the nearest integer.
LOG(double a)	double	Returns the natural logarithm (base e) of an expression.
LOG10(double a)	double	Returns the base 10 logarithm of an expression.
RANDOM( )	double	Returns a pseudo random number greater than or equal to 0.0 and less than 1.0.
ROUND(float a), ROUND(double a)	int, long	Returns the value of an expression rounded to an integer.
SIN(double a)	double	Returns the sine of an expression.
SINH(double a)	double	Returns the hyperbolic sine of an expression.
SQRT(double a)	double	Returns the positive square root of an expression.
TAN(double a)	double	Returns the tangent of an expression.
TANH(double a)	double	Returns the hyperbolic tangent of an expression.

Eval Functions

This group contains aggregate functions such as COUNT and SUM, along with useful utility methods such as IsEmpty.

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`AVG(col)`	`double`	Computes the average of the numeric values in a single column of a bag.
`CONCAT(String expression1, String expression2)` `CONCAT(byte[] expression1, byte[] expression2)`	`String, byte[]`	Concatenates two expressions of identical type.
`COUNT(DataBag bag)`	`long`	Computes the number of elements in a bag. Does not include null values.
`COUNT_STAR(DataBag bag)`	`long`	Computes the number of elements in a bag, including null values.
`DIFF(DataBag bag1, DataBag bag2)`	`DataBag`	Compares two bags. Any tuples that are in one bag but not the other are returned in a bag.
`IsEmpty(DataBag bag), IsEmpty(Map map)`	`boolean`	Checks if a bag or map is empty.
`MAX(col)`	`int, long, float, double`	Computes the maximum of the numeric values or chararrays in a single-column bag.
`MIN(col)`	`int, long, float, double`	Computes the minimum of the numeric values or chararrays in a single-column bag.
`DEFINE pluck PluckTuple(expression1) pluck(expression2)`	`Tuple`	Allows the user to specify a string prefix, and then filter for the columns in a relation that begin with that prefix.
`SIZE(expression)`	`long`	Computes the number of elements based on any Pig data type.
`SUBTRACT(DataBag bag1, DataBag bag2)`	`DataBag`	Returns bag composed of bag1 elements not in bag2.
`SUM(col)`	`double, long`	Computes the sum of the numeric values in a single-column bag.
`TOKENIZE(String expression [, 'field_delimiter'])`	`DataBag`	Splits a string and outputs a bag of words. If field_delimiter is null or not passed, the following will be used as delimiters: space [ ], double quote ["], comma [,] parenthesis [()], star [*]

String Functions

Providing convenient ways to handle text fields, these functions mirror commonly used String functions in other languages.

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`ENDSWITH(String string, String testAgainst)`	`boolean`	Tests inputs to determine if the first argument ends with the string in the second.
`EqualsIgnoreCase(String string1, String string2)`	`boolean`	Compares two strings ignoring case considerations.
`INDEXOF(String string, String 'character', int startIndex)`	`int`	Returns the index of the first occurrence of a character in a string, searching forward from a start index.
`LAST_INDEX_OF(String string, String 'character')`	`int`	Returns the index of the last occurrence of a character in a string, searching backward from the end of the string.
`LCFIRST(String expression)`	`String`	Converts the first character in a string to lower case.
`LOWER(String expression)`	`String`	Converts all characters in a string to lower case.
`LTRIM(String expression)`	`String`	Returns a copy of a string with only leading white space removed.
`REGEX_EXTRACT(String string, String regex, int index)`	`String`	Performs regular expression matching and extracts the matched group defined by an index parameter.
`REGEX_EXTRACT_ALL (String string, String regex)`	`Tuple`	Performs regular expression matching and extracts all matched groups.
`REPLACE(String string, String 'regExp', String 'newChar')`	`String`	Replaces existing characters in a string with new characters.
`RTRIM(String expression)`	`String`	Returns a copy of a string with only trailing white space removed.
`STARTSWITH(String string, String testAgainst)`	`boolean`	Tests inputs to determine if the first argument starts with the string in the second.
`STRSPLIT(String string, String regex, int limit)`	`Tuple`	Splits a string around matches of a given regular expression.
`SUBSTRING(String string, int startIndex, int stopIndex)`	`String`	Returns a substring from a given string.
`TRIM(String expression)`	`String`	Returns a copy of a string with leading and trailing white space removed.
`UCFIRST(String expression)`	`String`	Returns a string with the first character converted to upper case.
`UPPER(String expression)`	`String`	Returns a string converted to upper case.

DateTime Functions

Pig uses the Joda-Time DateTime class for date and time handling.

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`AddDuration(DateTime datetime, String duration)`	`DateTime`	Returns the result of a DateTime object plus an ISO 8601 duration string.
`CurrentTime()`	`DateTime`	Returns the DateTime object of the current time.
`DaysBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of days between two DateTime objects.
`GetDay(DateTime datetime)`	`int`	Returns the day of a month from a DateTime object.
`GetHour(DateTime datetime)`	`int`	Returns the hour of a day from a DateTime object.
`GetMilliSecond(DateTime datetime)`	`int`	Returns the millisecond of a second from a DateTime object.
`GetMinute(DateTime datetime)`	`int`	Returns the minute of an hour from a DateTime object.
`GetMonth(DateTime datetime)`	`int`	Returns the month of a year from a DateTime object.
`GetSecond(DateTime datetime)`	`int`	Returns the second of a minute from a DateTime object.
`GetWeek(DateTime datetime)`	`int`	Returns the week of a week year from a DateTime object.
`GetWeekYear(DateTime datetime)`	`int`	Returns the week year from a DateTime object.
`GetYear(DateTime datetime)`	`int`	Returns the year from a DateTime object.
`HoursBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of hours between two DateTime objects.
`MilliSecondsBetween(datetime1, datetime2)`	`long`	Returns the number of milliseconds between two DateTime objects.
`MinutesBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of minutes between two DateTime objects.
`MonthsBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of months between two DateTime objects.
`SecondsBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of seconds between two DateTime objects.
`SubtractDuration(DateTime datetime, String duration)`	`DateTime`	Returns the result of a DateTime object minus an ISO 8601 duration string.

DateTime Functions (cont...)

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`ToDate(long milliseconds)`	`DateTime`	Returns a DateTime object according to parameters.
`ToDate(String isostring)`	`DateTime`	Returns a DateTime object according to parameters.
`ToDate(String userstring, String format)`	`DateTime`	Returns a DateTime object according to parameters.
`ToDate(String userstring, String format, String timezone)`	`DateTime`	Returns a DateTime object according to parameters.
`ToMilliSeconds(DateTime datetime)`	`long`	Returns the number of milliseconds elapsed since January 1, 1970, 00:00:00.000 GMT for a DateTime object.
`ToString(DateTime datetime)`	`String`	Converts the DateTime object to the ISO string.
`ToString(DateTime datetime, String format)`	`String`	Converts the DateTime object to the customized string.
`ToUnixTime(DateTime datetime)`	`long`	Returns the Unix Time as long for a DateTime object. Unix Time is the number of seconds elapsed since January 1, 1970, 00:00:00.000 GMT.
`WeeksBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of weeks between two DateTime objects.
`YearsBetween(DateTime datetime1, DateTime datetime2)`	`long`	Returns the number of years between two DateTime objects.

Bag and Tuple Functions

One of the things that makes Pig powerful is its use of complex data types. These functions exist to manipulate data stored in bags, tuples, and maps.

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`TOTUPLE(expression [, expression ...])`	`Tuple`	Converts one or more expressions to type tuple.
`TOBAG(expression [, expression ...])`	`DataBag`	Converts one or more expressions to type bag.
`TOMAP(key-expression, value-expression [, key-expression, value-expression ...])`	`Map`	Converts key/value expression pairs into a map.
`TOP(int topN, column, relation)`	`DataBag`	Returns the top-n tuples from a bag of tuples.

Load/Store Functions

Crucial to data manipulation in Pig, load and store functions bring data into Pig and push it out again in a multitude of formats.

NAME	FILE FORMAT	LOAD/STORE	
`org.apache.pig.piggybank.storage.avro.AvroStorage()` `org.apache.pig.piggybank.storage.avro.AvroStorage('no_schema_check', 'SCHEMA_FILE', 'PATH');`	Avro	Load, Store	
`org.apache.pig.piggybank.storage.apachelog.CommonLogLoader()`	Common Log Format, Combined Log Format	Load	
`org.apache.pig.piggybank.storage.CSVExcelStorage()` `org.apache.pig.piggybank.storage.CSVExcelStorage('DELIMITER', 'YES_MULTILINE', 'NOCHANGE', 'SKIP_INPUT_HEADER')`	CSV	Load, Store	
`com.mortardata.pig.storage.DynamoDBStorage('DYNAMODB_TABLE', 'DYNAMODB_AWS_ACCESS_KEY_ID', 'DYNAMODB_AWS_SECRET_ACCESS_KEY')`	DynamoDB	Store	
see PostgreSQL	Greenplum	Store	
`org.apache.pig.piggybank.storage.FixedWidthLoader('-5, 7-10, 10-18')` `org.apache.pig.piggybank.storage.FixedWidthLoader('-5, 7-10, 10-18', 'SKIP_HEADER', 'SCHEMA')`	Fixed-Width	Load, Store	
`org.apache.pig.piggybank.storage.HiveColumnarLoader('SCHEMA')`	Hive	Load	
`org.apache.pig.piggybank.storage.JsonLoader()` `org.apache.pig.piggybank.storage.JsonLoader('SCHEMA')`	JSON	Load, Store	
`com.mongodb.hadoop.pig.MongoLoader()` `com.mongodb.hadoop.pig.MongoLoader('SCHEMA')`	MongoDB	Load, Store	
`com.mortardata.pig.PapertrailLoader()`	Papertrail Logs	Load	
`PigStorage()` `PigStorage('DELIMITER')`	character-delimited (CSV, TSV)	Load, Store	
`org.apache.pig.piggybank.storage.DBStorage('org.postgresql.Driver', 'jdbc:postgresql://$POSTGRESQL_HOST/$POSTGRESQL_DATABASE', 'POSTGRESQL_USER', 'POSTGRESQL_PASS', 'INSERT INTO my_table(my_col_1,my_col_2,my_col_3) VALUES (?,?,?)')`	PostgreSQL	Store	
`TextLoader()`	Text/Unformatted	Load	
`org.apache.pig.piggybank.storage.StreamingXMLLoader('DOCUMENT')`	XML	Load	

SQL -> Pig

Because many people come to Pig from a relational database background, we've included a handy translation from SQL concepts to their Pig equivalents.

SQL FUNCTION	SQL	PIG
SELECT	SELECT column_name,column_name FROM table_name;	FOREACH alias GENERATE column_name, column_name;
SELECT *	SELECT * FROM table_name;	FOREACH alias GENERATE *;
DISTINCT	SELECT DISTINCT column_name,column_name FROM table_name;	DISTINCT(FOREACH alias GENERATE column_name, column_name);
WHERE	SELECT column_name,column_name FROM table_name WHERE column_name operator value;	FOREACH (FILTER alias BY column_name operator value) GENERATE column_name, column_name;
AND/OR	... WHERE (column_name operator value1 AND column_name operator value2) OR column_name operator value3;	FILTER alias BY (column_name operator value1 AND column_name operator value2) OR column_name operator value3;
ORDER BY	... ORDER BY column_name ASC|DESC, column_name ASC|DESC;	ORDER alias BY column_name ASC|DESC, column_name ASC|DESC;
TOP/LIMIT	SELECT TOP number column_name FROM table_name ORDER BY column_name ASC|DESC; SELECT column_name FROM table_name ORDER BY column_name ASC|DESC LIMIT number;	FOREACH (GROUP alias BY column_name) GENERATE LIMIT alias number; TOP(number, column_index, alias);
GROUP BY	SELECT function(column_name) FROM table GROUP BY column_name;	FOREACH (GROUP alias BY column_name) GENERATE function(alias.column_name);
LIKE	... WHERE column_name LIKE pattern;	FILTER alias BY REGEX_EXTRACT(column_name, pattern, 1) IS NOT NULL;
IN	... WHERE column_name IN (value1,value2,...);	FILTER alias BY column_name IN (value1, value2,...);
JOIN	SELECT column_name(s) FROM table1 JOIN table2 ON table1.column_name=table2.column_name;	FOREACH (JOIN alias1 BY column_name, alias2 BY column_name) GENERATE column_name(s);

SQL -> Pig (cont...)

SQL FUNCTION	SQL	PIG
LEFT/RIGHT/FULL OUTER JOIN	SELECT column_name(s) FROM table1 LEFT|RIGHT|FULL OUTER JOIN table2 ON table1.column_name=table2.column_name;	FOREACH (JOIN alias1 BY column_name LEFT|RIGHT|FULL, alias2 BY column_name) GENERATE column_name(s);
UNION ALL	SELECT column_name(s) FROM table1 UNION ALL SELECT column_name(s) FROM table2;	UNION alias1, alias2;
AVG	SELECT AVG(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE AVG(alias.column_name);
COUNT	SELECT COUNT(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE COUNT(alias);

COUNT DISTINCT	SELECT COUNT(DISTINCT column_name) FROM table_name;	FOREACH alias {
		unique_column = DISTINCT column_name;
		GENERATE COUNT(unique_column);
		};

MAX	SELECT MAX(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE MAX(alias.column_name);
MIN	SELECT MIN(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE MIN(alias.column_name);
SUM	SELECT SUM(column_name) FROM table_name;	FOREACH (GROUP alias ALL) GENERATE SUM(alias.column_name);
HAVING	... HAVING aggregate_function(column_name) operator value;	FILTER alias BY aggregate_function(column_name) operator value;
UCASE/UPPER	SELECT UCASE(column_name) FROM table_name;	FOREACH alias GENERATE UPPER(column_name);
LCASE/LOWER	SELECT LCASE(column_name) FROM table_name;	FOREACH alias GENERATE LOWER(column_name);
SUBSTRING	SELECT SUBSTRING(column_name,start,length) AS some_name FROM table_name;	FOREACH alias GENERATE SUBSTRING(column_name, start, start+length) as some_name;
LEN	SELECT LEN(column_name) FROM table_name;	FOREACH alias GENERATE SIZE(column_name);
ROUND	SELECT ROUND(column_name,0) FROM table_name;	FOREACH alias GENERATE ROUND(column_name);