Pig Latin Cheatsheet

Pig Latin keywords are case-insensitive, but are often written in all caps to distinguish them from other elements.

AS

Assign a schema to a value. The value may be a bag, a tuple, or a primitive type.

```
b = LOAD 'file' AS (a:int, b:chararray);
c = FOREACH b GENERATE $1 AS value;
```

COGROUP†

Synonymous with GROUP, but usually used when multiple relations are used.

```
c = COGROUP a BY name, b BY manager;
```

CROSS†

Generate the cross product of two relations.

```
c = CROSS a, b;
```

DEFINE

Assign an alias for a UDF or streaming command.

```
DEFINE myFunc com.mycompany.LongName('arg');
```

DISTINCT†

Filter out any duplicate records in the supplied relation.

```
b = DISTINCT a;
```

FILTER/BY

Filter out records in a relation based on a predicate.

```
b = FILTER a BY name == 'my_name';
```

FLATTEN

FLATTEN removes levels of nesting. For example, flattening a tuple produces the fields contained in the tuple. The effect is different depending on the input type.

FOREACH/GENERATE

Generate a new relation based on records

GROUP/BY†

Generate a new data set grouped by the specified field. It should be noted that the schema of the resulting data set will be {group:name_schema, {source:full_schema}}.

```
b = GROUP source BY name;
```

JOIN†

Join two relations by a common field. There are lots of ways to use JOIN depending on the expected inputs. Check a more complete reference for details.

```
c = JOIN a BY id, b BY friend_id;
```

LOAD

The load keyword specifies a path from which to load data. The default is to load data using PigStorage(',').

```
a = LOAD '/path/to/data/';
```

ORDER†

Sort a relation by one or more fields.

```
b = ORDER a BY name;
```

REGISTER

Allow UDFs from the specified JAR file to be used. Additional jars can also be specified on the command line by setting the pig.additional.jars property.

```
REGISTER 'my.jar';
```

SET

Set a property for the duration of the script.

```
SET job.name 'My awesome job';
```

SPLIT/INTO/IF

Copy the elements of a relation into other relations based on a set of conditions. Note that an element may be inserted into several of the output relations if multiple conditions are true.

```
SPLIT a INTO b IF name == 'my_name',
  c IF address MATCHES '.*Chicago, IL.*';
```

STORE/INTO

Stores a relation to the filesystem.

```
STORE a INTO '/my/output/dir/';
```

TOKENIZE

Split a string into a bag containing all of the words in the string. Tokenize is an example of an Eval function.

```
b = FOREACH a GENERATE TOKENIZE(a.text);
```

UNION

Combine two relations into a single relation.

```
c = UNION a, b;
```

USING

Syntactic sugar for several other commands.

```
a = LOAD b USING PigStorage('|');
c = JOIN a BY name, b BY name USING 'skewed';
b = GROUP a BY name USING 'collected';
```

a = LOAD '/mydata/' USING PigStorage(',') AS (name:chararray, address:chararray);

Create a relation by reading the path via PigStorage. Each record has two fields, name and address. Any fields beyond those defined in the schema are discarded.

b = GROUP a BY name;

c = FOREACH b GENERATE group, COUNT(a);

Create a new relation containing the grouping element (name) and the count of records in each group.

STORE c INTO '/myoutput/' USING JsonStorage();

Write the data out to the filesystem in JSON format.

†Forces a reduce phase

Hive Cheatsheet

Hive allows unstructured and semi-structured data to be queried using MapReduce via a SQL-like syntax. Since the point of using Hive is to allow those familiar with SQL to run queries without learning MapReduce programming, I won't explain all of the details of HQL here, but here are some of the higher-level concepts you might run into.

Managed (Internal) Table A table managed by hive. When data is added to the table, it is moved out of the filesystem, and when the table is dropped, all data is deleted.

External Table A table whose data is not managed by Hive, but lives in the filesystem. Multiple external tables can refer to the same data.

Metastore The Hive Metastore is a repository for metadata, i.e. table definitions. The default metastore uses the Derby in-memory database, but Hive can be configured to point to any JDBC-compliant database.

Partitions Partitions are a way of optimizing queries which commonly specify a where clause based on a certain subset of columns. Since Hive tables are actually stored in the filesystem, partitions are implemented as directories, which reduces storage cost for the partition keys. Partitions for external tables must be configured manually.

The Hive Metastore can be run in three distinct modes of operation, with different implications.

Embedded In embedded mode, the client, server, and the metastore all run in the same process. The metastore uses the Derby database to store metadata.

Local In local mode, the client and server still run in the same process, but the metastore is a remote database. This requires that the client have a valid JDBC connection string for the metastore, and any queries are run using the logged-in user's credentials and permissions.

Standalone/Remote In this case the server is run in a separate process from both the client and the metastore. This is often used when control over the metastore connection details is desired.

Java Map/Reduce Cheatsheet

Several of the interfaces and classes used in direct Java MapReduce programming are useful to developers Pig. When the built-in storage formats and evaluation functions are not sufficient to accomplish a task, custom Java code can be easily built to integrate with Pig. All of these classes should be assumed to be in the org.apache.hadoop.mapreduce package rather than the older org.apache.hadoop.mapred package.[1]

org.apache.hadoop.mapreduce.RecordReader<K,V>
The RecordReader class takes an InputSplit and returns records. Some examples are the LineRecordReader, which reads one line at a time, or the SequenceFileRecordReader, which reads binary sequencefiles.

org.apache.hadoop.mapreduce.InputFormat<K,V>
The InputFormat determines how the data required for a job will be returned. This is done both by providing the list of splits to be used and the RecordReader to parse those splits into records.

org.apache.pig.LoadFunc
Each LoadFunc has an associated InputFormat (see above). The LoadFunc's getNext() method takes each record from the RecordReader provided by its InputFormat and returns a Tuple.

org.apache.pig.LoadMetadata
This interface is optional, and among other things, returns a schema for the Tuples provided by getNext(). This allows you to use the data returned by the LoadFunc without specifing 'AS (...)';

org.apache.hadoop.mapreduce.RecordWriter<K,V>
Responsible for taking key/value pairs as input and writing them to the filesystem.

org.apache.hadoop.mapreduce.OutputFormat<K,V>
Responsible for providing a RecordWriter to store records.

org.apache.pig.StoreFunc
The StoreFunc and its subclasses take records from Pig and write them to the filesystem (or somewhere else). The putNext() method is called for each Tuple. Typically storage is done through an OutputFormat.

[1]Yes. This is terribly confusing.