UNIT- VIII

Introducing Pig

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

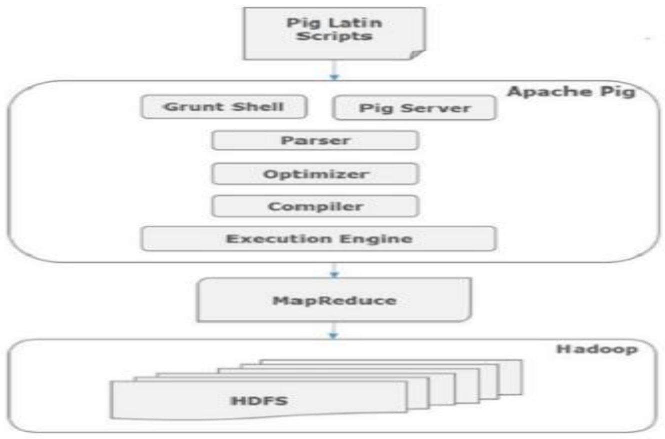
History and Anatomy of Pig:

In 2006, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset. In 2007, Apache Pig was open sourced via Apache incubator. In 2008, the first release of Apache Pig came out. In 2010, Apache Pig graduated as an Apache top-level project.

The language used to analyze data in Hadoop using Pig is known as Pig Latin. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.



Pig on Hadoop:

Listed below are the major differences between Apache Pig and MapReduce.

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

Listed below are the major differences between Apache Pig and SQL.

Pig	SQL
Pig Latin is a procedural language.	SQL is a declarative language.
In Apache Pig, schema is optional. We can store data without designing a schema (values are stored as $01, $02 etc.)	Schema is mandatory in SQL.
The data model in Apache Pig is nested relational.	The data model used in SQL is flat relational.
Apache Pig provides limited opportunity for Query optimization.	There is more opportunity for query optimization in SQL.

In addition to above differences, Apache Pig Latin −

- Allows splits in the pipeline.
- Allows developers to store data anywhere in the pipeline.
- Declares execution plans.
- Provides operators to perform ETL (Extract, Transform, and Load) functions.

Pig Features,

Apache Pig comes with the following features −

- Rich set of operators − It provides many operators to perform operations like join, sort, filer, etc.
- Ease of programming − Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- Optimization opportunities − The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
- Extensibility − Using the existing operators, users can develop their own functions to read, process, and write data.
- UDF's − Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.
- Handles all kinds of data − Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Pig Philosophy :

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

- Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Word count example using Pig :

1. Pig Word Count Scripts

```
A = load './input.txt';

B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;

C = group B by word;

D = foreach C generate COUNT(B), group;

store D into './wordcount';
```

2. Running Pig Word Count Script

   local mode:

```
bin/pig -x local wordcount.pig
```

   mapreduce mode:

```
hadoop dfs -copyFromLocal input.txt input/input.txt

bin/pig -x mapreduce wordcount.pig
```

3. Programming Word Count using Pig

   Here are major steps to develop Pig word count application.

   Loads data from the file system.
   LOAD 'data' [USING function] [AS schema];

   Sample:

   records = load 'student.txt' as (name:chararray, age:int, gpa:double);
   Generates data transformations based on columns of data.
   alias = FOREACH { gen_blk | nested_gen_blk } [AS schema];
   Sample:

   words = foreach lins generate flatten(TOKENIZE((chararray)$0)) as word;
   Sometimes we want to eliminate nesting. This can be accomplished via the FLATTEN keyword.
   words = foreach lines generate flatten(TOKENIZE((chararray)$0)) as word;
   The GROUP operator groups together tuples that have the same group key (key field).

```
alias = GROUP alias { ALL | BY expression} [, alias ALL | BY expression ...]
[USING 'collected'] [PARALLEL n];
```

Sample:

```
word_groups = group words by word;
```

Use the COUNT function to compute the number of elements in a bag.

```
COUNT(expression)
```

Sample:

```
D = foreach C generate COUNT(B), group;
```

The above program steps will generate parallel executable tasks which can be distributed across multiple machines in a Hadoop cluster to count the number of words in a text file.

Use Case for Pig:

Time Sensitive Data Loads

Loading data is a key part of many businesses. Data comes in from outside of the database in text, XML, CSV, or some other arbitrary file format. The data then has to be processed into a different formats and loaded into a database for later querying. Sometimes there are a lot of steps involved, sometimes the data has to be translated into an intermediate format, but most of the time it gets into the database, some failure it to be expected, right?

Loading large volumes of data can become a problem as the volume of data increases: the more data there is, the longer it takes to load. To get around this problem people routinely buy bigger, faster servers and use more fast disks. There comes a point when you can’t add more CPUs or RAM to a server and increasing the I/O capacity won’t help. Parallelizing ETL processes can be hard even on one machine, much less scaling ETL out across several machines.

Pig is built on top of Hadoop, so it’s able to scale across multiple processors and servers which makes it easy to processes massive data sets. Many ETL processes lend themselves to being decomposed into manageable chunks; Pig is no exception. Pig builds MapReduce jobs behind the scenes to spread load across many servers. By taking advantage of the simple building blocks of Hadoop, data professionals are able to build simple, easily understood scripts to process and analyze massive quantities of data in a massively parallel environment.

Parallel rockets make a single pig faster

An advantage of being able to scale out across many servers is that doubling throughput is often as easy as doubling the number of servers working on a problem.

If one server can solve a problem in 12 hours, 24 servers should be able to solve it in 30 minutes.

Pig Primitive Data Types:

Given below table describes the Pig Latin data types.

S.N.	Data Type	Description & Example
1	int	Represents a signed 32-bit integer. Example : 8
2	long	Represents a signed 64-bit integer. Example : 5L
3	float	Represents a signed 32-bit floating point. Example : 5.5F
4	double	Represents a 64-bit floating point. Example : 10.5
5	chararray	Represents a character array (string) in Unicode UTF-8 format. Example : 'tutorials point'
6	Bytearray	Represents a Byte array (blob).
7	Boolean	Represents a Boolean value. Example : true/ false.
8	Datetime	Represents a date-time. Example : 1970-01-01T00:00:00.000+00:00
9	Biginteger	Represents a Java BigInteger. Example : 60708090709
10	Bigdecimal	Represents a Java BigDecimal Example : 185.98376256272893883
		Complex Types
11	Tuple	A tuple is an ordered set of fields. Example : (raja, 30)
12	Bag	A bag is a collection of tuples. Example : {(raju,30),(Mohhammad,45)}

13	Map	A Map is a set of key-value pairs. Example : ['name'#'Raju', 'age'#30]

Colletion Types and NULL :

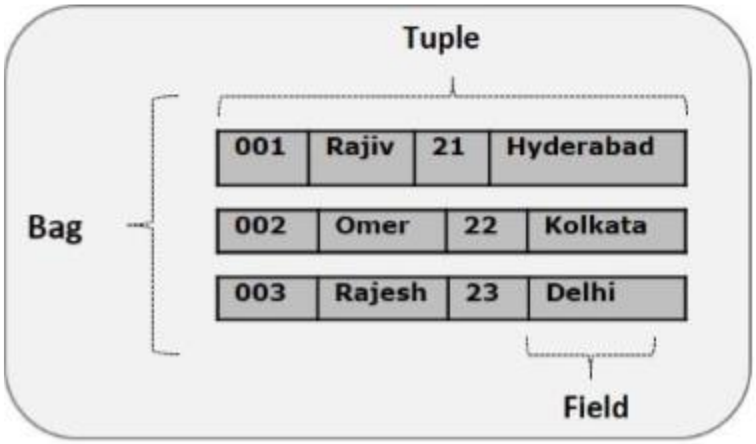
Values for all the above data types can be NULL. Apache Pig treats null values in a similar way as SQL does.

A null can be an unknown value or a non-existent value. It is used as a placeholder for optional values. These nulls can occur naturally or can be the result of an operation.

Pig Latin Overview :

Pig Latin Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as map and tuple. Given below is the diagrammatical representation of Pig Latin's data model.



Atom

Any single value in Pig Latin, irrespective of their data, type is known as an Atom. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a field.

Example − 'raja' or '30'

Pig Latin Grammar – Comments:

The name of a file containing one or more parameters.

A parameter file will contain one line per parameter. Empty lines are allowed. Perl-style (#) comment lines are also allowed. Comments must take a full line and # must be the first character on the line. Each parameter line will be of the form: param_name = param_value. White spaces around = are allowed but are optional.

Keywords :

Pig reserved keywords are listed here.

-- A	and, any, all, arrange, as, asc, AVG
-- B	bag, BinStorage, by, bytearray
-- C	cache, cat, cd, chararray, cogroup, CONCAT, copyFromLocal, copyToLocal, COUNT, cp, cross
-- D	%declare, %default, define, desc, describe, DIFF, distinct, double, du, dump
-- E	e, E, eval, exec, explain
-- F	f, F, filter, flatten, float, foreach, full
-- G	generate, group
-- H	help
-- I	if, illustrate, import, inner, input, int, into, is
-- J	join
-- K	kill
-- L	l, L, left, limit, load, long, ls
-- M	map, matches, MAX, MIN, mkdir, mv
-- N	not, null
-- O	onschema, or, order, outer, output
-- P	parallel, pig, PigDump, PigStorage, pwd
-- Q	quit
-- R	register, right, rm, rmf, run
-- S	sample, set, ship, SIZE, split, stderr, stdin, stdout, store, stream, SUM
-- T	TextLoader, TOKENIZE, through, tuple
-- U	union, using
-- V, W, X,	

Y, Z

Identifiers :

dentifiers include the names of relations (aliases), fields, variables, and so on. In Pig, identifiers start with a letter and can be followed by any number of letters, digits, or underscores.

Valid identifiers:

```
A
A123
abc_123_BeX_
```

Invalid identifiers:

```
_A123
abc_$
A!B
```

Case sensitivity in Pig:

The names (aliases) of relations and fields are case sensitive. The names of Pig Latin functions are case sensitive. The names of parameters (see Parameter Substitution) and all other Pig Latin keywords (see Reserved Keywords) are case insensitive.

In the example below, note the following:

- The names (aliases) of relations A, B, and C are case sensitive.
- The names (aliases) of fields f1, f2, and f3 are case sensitive.
- Function names PigStorage and COUNT are case sensitive.
- Keywords LOAD, USING, AS, GROUP, BY, FOREACH, GENERATE, and DUMP are case insensitive. They can also be written as load, using, as, group, by, etc.
- In the FOREACH statement, the field in relation B is referred to by positional notation ($0).

```
grunt> A = LOAD 'data' USING PigStorage() AS (f1:int, f2:int, f3:int);
grunt> B = GROUP A BY f1;
grunt> C = FOREACH B GENERATE COUNT ($0);
grunt> DUMP C;
```

Common Operators in Pig:

The following table describes the arithmetic operators of Pig Latin. Suppose a = 10 and b = 20.

Operator	Description	Example
+	Addition − Adds values on either side of the operator	a + b will give 30
−	Subtraction − Subtracts right hand operand from left hand operand	a − b will give −10
*	Multiplication − Multiplies values on either side of the operator	a * b will give 200
/	Division − Divides left hand operand by right hand operand	b / a will give 2
%	Modulus − Divides left hand operand by right hand operand and returns remainder	b % a will give 0
? :	Bincond − Evaluates the Boolean operators. It has three operands as shown below. variable x = (expression) ? value1 *if true* : value2 *if false*.	b = (a == 1)? 20: 30; if a = 1 the value of b is 20. if a!=1 the value of b is 30.

CASE
 	
WHEN
 	
THEN
 	
ELSE END
	Case − The case operator is equivalent to nested bincond operator.	CASE f2 % 2
		
		WHEN 0 THEN 'even'
		
		WHEN 1 THEN 'odd'
		
		END

Pig Statements – LOAD :

You can load data into Apache Pig from the file system (HDFS/ Local) using LOAD operator of Pig Latin.

Syntax

The load statement consists of two parts divided by the "=" operator. On the left-hand side, we need to mention the name of the relation where we want to store the data, and on the right-hand side, we have to define how we store the data. Given below is the syntax of the Load operator.

Relation_name = LOAD 'Input file path' USING function as schema;

Where,

- relation_name − We have to mention the relation in which we want to store the data.

- Input file path − We have to mention the HDFS directory where the file is stored. (In MapReduce mode)
- function − We have to choose a function from the set of load functions provided by Apache Pig (BinStorage, JsonLoader, PigStorage, TextLoader).
- Schema − We have to define the schema of the data. We can define the required schema as follows −

(column1 : data type, column2 : data type, column3 : data type);

Note − We load the data without specifying the schema. In that case, the columns will be addressed as $01, $02, etc... (check).

Example

As an example, let us load the data in student_data.txt in Pig under the schema named Student using the LOAD command.

STORE:

Execute the Load Statement

Now load the data from the file student_data.txt into Pig by executing the following Pig Latin statement in the Grunt shell.

```
grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student_data.txt'
   USING PigStorage(',')
   as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,
   city:chararray );
```

Following is the description of the above statement.

Relation name	We have stored the data in the schema student.
Input file path	We are reading data from the file student_data.txt, which is in the /pig_data/ directory of HDFS.
Storage function	We have used the PigStorage() function. It loads and stores data as structured text files. It takes a delimiter using which each entity of a tuple is separated, as a parameter. By default, it takes '\t' as a parameter.
	We have stored the data using the following schema.
schema	column id firstname lastname phone city datatype int char array char array char array char array

Note − The load statement will simply load the data into the specified relation in Pig. To verify the execution of the Load statement, you have to use the Diagnostic Operators which are discussed in the next chapters.

DUMP:

Dump Operator

The Dump operator is used to run the Pig Latin statements and display the results on the screen. It is generally used for debugging Purpose.

Syntax

Given below is the syntax of the Dump operator.

```
grunt> Dump Relation_Name
```

Example

Assume we have a file student_data.txt in HDFS with the following content.

```
001,Rajiv,Reddy,9848022337,Hyderabad
002,siddarth,Battacharya,9848022338,Kolkata
003,Rajesh,Khanna,9848022339,Delhi
004,Preethi,Agarwal,9848022330,Pune
005,Trupthi,Mohanthy,9848022336,Bhuwaneshwar
006,Archana,Mishra,9848022335,Chennai.
```

And we have read it into a relation student using the LOAD operator as shown below.

```
grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student_data.txt'
   USING PigStorage(',')
   as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,
   city:chararray );
```

Now, let us print the contents of the relation using the Dump operator as shown below.

```
grunt> Dump student
```

Once you execute the above Pig Latin statement, it will start a MapReduce job to read data from HDFS. It will produce the following output.

```
2015-10-01 15:05:27,642 [main]
INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLaunch
er -
100% complete
```

```
2015-10-01 15:05:27,652 [main]
INFO  org.apache.pig.tools.pigstats.mapreduce.SimplePigStats - Script Statistics:
HadoopVersion  PigVersion  UserId    StartedAt            FinishedAt       Features
2.6.0         0.15.0      Hadoop  2015-10-01 15:03:11  2015-10-01 05:27     UNKNOWN

Success!
Job Stats (time in seconds):

JobId          job_14459_0004
Maps                1
Reduces              0
MaxMapTime          n/a
MinMapTime          n/a
AvgMapTime          n/a
MedianMapTime       n/a
MaxReduceTime        0
MinReduceTime        0
AvgReduceTime        0
MedianReducetime     0
Alias            student
Feature           MAP_ONLY
Outputs           hdfs://localhost:9000/tmp/temp580182027/tmp757878456,

Input(s): Successfully read 0 records from: "hdfs://localhost:9000/pig_data/
student_data.txt"

Output(s): Successfully stored 0 records in:
"hdfs://localhost:9000/tmp/temp580182027/
tmp757878456"

Counters: Total records written : 0 Total bytes written : 0 Spillable Memory Manager
spill count : 0Total bags proactively spilled: 0 Total records proactively spilled: 0

Job DAG: job_1443519499159_0004

2015-10-01 15:06:28,403 [main]
INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLau
ncher - Success!
2015-10-01 15:06:28,441 [main] INFO  org.apache.pig.data.SchemaTupleBackend -
Key [pig.schematuple] was not set... will not generate code.
2015-10-01 15:06:28,485 [main]
INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths
to process : 1
2015-10-01 15:06:28,485 [main]
INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input
paths
to process : 1

(1,Rajiv,Reddy,9848022337,Hyderabad)
```

```
(2,siddarth,Battacharya,9848022338,Kolkata)
(3,Rajesh,Khanna,9848022339,Delhi)
(4,Preethi,Agarwal,9848022330,Pune)
(5,Trupthi,Mohanthy,9848022336,Bhuwaneshwar)
(6,Archana,Mishra,9848022335,Chennai)
```

Interactive Shell – GRUNT:

After invoking the Grunt shell, you can run your Pig scripts in the shell. In addition to that, there are certain useful shell and utility commands provided by the Grunt shell. This chapter explains the shell and utility commands provided by the Grunt shell.

Note − In some portions of this chapter, the commands like Load and Store are used. Refer the respective chapters to get in-detail information on them.

Shell Commands

The Grunt shell of Apache Pig is mainly used to write Pig Latin scripts. Prior to that, we can invoke any shell commands using sh and fs.

sh Command

Using sh command, we can invoke any shell commands from the Grunt shell. Using sh command from the Grunt shell, we cannot execute the commands that are a part of the shell environment (ex − cd).

Syntax

Given below is the syntax of sh command.

```
grunt> sh shell command parameters
```

Example

We can invoke the ls command of Linux shell from the Grunt shell using the sh option as shown below. In this example, it lists out the files in the /pig/bin/ directory.

```
grunt> sh ls

pig
pig_1444799121955.log
pig.cmd
pig.py
```

fs Command

Using the fs command, we can invoke any FsShell commands from the Grunt shell.

Syntax

Given below is the syntax of fs command.

```
grunt> sh File System command parameters
```

Example

We can invoke the ls command of HDFS from the Grunt shell using fs command. In the following example, it lists the files in the HDFS root directory.

```
grunt> fs –ls
Found 3 items
drwxrwxrwx   - Hadoop supergroup          0 2015-09-08 14:13 Hbase
drwxr-xr-x   - Hadoop supergroup          0 2015-09-09 14:52 seqgen_data
drwxr-xr-x   - Hadoop supergroup          0 2015-09-08 11:30 twitter_data
```

In the same way, we can invoke all the other file system shell commands from the Grunt shell using the fs command.

FILTER:

The FILTER operator is used to select the required tuples from a relation based on a condition.

Syntax

Given below is the syntax of the FILTER operator.

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' 
USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, 
city:chararray);
```

Let us now use the Filter operator to get the details of the students who belong to the city Chennai.

```
filter_data = FILTER student_details BY city == 'Chennai';
```

Verification

Verify the relation filter_data using the DUMP operator as shown below.

```
grunt> Dump filter_data;
```

SORT:

In the previous chapter, we learnt how to load data into Apache Pig. You can store the loaded data in the file system using the store operator. This chapter explains how to store data in Apache Pig using the Store operator.

Syntax

Given below is the syntax of the Store statement.

```
STORE Relation_name INTO ' required_directory_path ' [USING function];
```

Example

Assume we have a file student_data.txt in HDFS with the following content.

```
001,Rajiv,Reddy,9848022337,Hyderabad
002,siddarth,Battacharya,9848022338,Kolkata
003,Rajesh,Khanna,9848022339,Delhi
004,Preethi,Agarwal,9848022330,Pune
005,Trupthi,Mohanthy,9848022336,Bhuwaneshwar
006,Archana,Mishra,9848022335,Chennai.
```

And we have read it into a relation student using the LOAD operator as shown below.

```
grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student_data.txt'
   USING PigStorage(',')
   as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,
   city:chararray );
```

Now, let us store the relation in the HDFS directory “/pig_Output/” as shown below.

```
grunt> STORE student INTO ' hdfs://localhost:9000/pig_Output/ ' USING PigStorage
(','
```

GROUP BY:

The GROUP operator is used to group the data in one or more relations. It collects the data having the same key.

Syntax

Given below is the syntax of the group operator.

```
grunt> Group_data = GROUP Relation_name BY age;
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Apache Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt'
USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray,
city:chararray);
```

Now, let us group the records/tuples in the relation by age as shown below.

```
grunt> group_data = GROUP student_details by age;
```

Verification

Verify the relation group_data using the DUMP operator as shown below.

```
grunt> Dump group_data;
```

ORDER BY:

The ORDER BY operator is used to display the contents of a relation in a sorted order based on one or more fields.

Syntax

Given below is the syntax of the ORDER BY operator.

```
grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt'
USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray,
city:chararray);
```

Let us now sort the relation in a descending order based on the age of the student and store it into another relation named order_by_data using the ORDER BY operator as shown below.

```
grunt> order_by_data = ORDER student_details BY age DESC;
```

Verification

Verify the relation order_by_data using the DUMP operator as shown below.

```
grunt> Dump order_by_data;
```

JOIN:

The JOIN operator is used to combine records from two or more relations. While performing a join operation, we declare one (or a group of) tuple(s) from each relation, as keys. When these keys match, the two particular tuples are matched, else the records are dropped. Joins can be of the following types −

- Self-join
- Inner-join
- Outer-join − left join, right join, and full join

This chapter explains with examples how to use the join operator in Pig Latin. Assume that we have two files namely customers.txt and orders.txt in the /pig_data/ directory of HDFS as shown below.

customers.txt

```
1,Ramesh,32,Ahmedabad,2000.00
2,Khilan,25,Delhi,1500.00
3,kaushik,23,Kota,2000.00
4,Chaitali,25,Mumbai,6500.00
5,Hardik,27,Bhopal,8500.00
6,Komal,22,MP,4500.00
7,Muffy,24,Indore,10000.00
```

orders.txt

```
102,2009-10-08 00:00:00,3,3000
100,2009-10-08 00:00:00,3,1500
101,2009-11-20 00:00:00,2,1560
103,2008-05-20 00:00:00,4,2060
```

And we have loaded these two files into Pig with the relations customers and orders as shown below.

```
grunt> customers = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING
PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);

grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING
PigStorage(',')
```

```
   as (oid:int, date:chararray, customer_id:int, amount:int);
```

LIMIT:

The LIMIT operator is used to get a limited number of tuples from a relation.

Syntax

Given below is the syntax of the LIMIT operator.

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt'
USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray,
city:chararray);
```

Now, let’s sort the relation in descending order based on the age of the student and store it into another relation named limit_data using the ORDER BY operator as shown below.

```
grunt> limit_data = LIMIT student_details 4;
```

Verification

Verify the relation limit_data using the DUMP operator as shown below.

```
grunt> Dump limit_data;
```

Output

It will produce the following output, displaying the contents of the relation limit_data as follows.

```
(1,Rajiv,Reddy,21,9848022337,Hyderabad)
(2,siddarth,Battacharya,22,9848022338,Kolkata)
(3,Rajesh,Khanna,22,9848022339,Delhi)
(4,Preethi,Agarwal,21,9848022330,Pune)
```

Pig Latin Script - Local Mode Map Reduce Mode

```
           Local mode                         MapReduce mode
$ pig -x local Sample_script.pig $ pig -x mapreduce Sample_script.pig
```

You can execute it from the Grunt shell as well using the exec command as shown below.

```
grunt> exec /sample_script.pig
```

Running Pig Script Working with Field:

Here in this chapter, we will see how how to run Apache Pig scripts in batch mode.

Comments in Pig Script

While writing a script in a file, we can include comments in it as shown below.

Multi-line comments

We will begin the multi-line comments with '/*', end them with '*/'.

```
/* These are the multi-line comments
  In the pig script */
```

Single –line comments

We will begin the single-line comments with '--'.

```
--we can write single line comments like this.
```

Executing Pig Script in Batch mode

While executing Apache Pig statements in batch mode, follow the steps given below.

Step 1

Write all the required Pig Latin statements in a single file. We can write all the Pig Latin statements and commands in a single file and save it as .pig file.

Step 2

Execute the Apache Pig script. You can execute the Pig script from the shell (Linux) as shown below.

Tuple,

<u>AVG()</u> To compute the average of the numerical values within a bag.	
2	<u>BagToString()</u> To concatenate the elements of a bag into a string. While concatenating, we can place a delimiter between these values (optional).
3	<u>CONCAT()</u> To concatenate two or more expressions of same type.
4	<u>COUNT()</u> To get the number of elements in a bag, while counting the number of tuples in a bag.
5	<u>COUNT_STAR()</u> It is similar to the COUNT() function. It is used to get the number of elements in a bag.

Bag User Defined Function:

Given below is the list of Bag functions.

S.N.	Function & Description
1	<u>TOBAG()</u> To convert two or more expressions into a bag.
2	<u>TOP()</u>

To get the top N tuples of a relation.

3 TOTUPLE()

To convert one or more expressions into a tuple.

4 TOMAP()

To convert the key-value pairs into a Map.

Parameters in Pig:

When running Pig in a production environment, you'll likely have one or more Pig Latin scripts that run on a recurring basis (daily, weekly, monthly, etc.) that need to locate their input data based on when or where they are run. For example, you may have a Pig job that performs daily log ingestion by geographic region. It would be costly and error prone to manually edit the script to reference the location of the input data each time log data needs to be ingested. Ideally, you'd like to pass the date and geographic region to the Pig script as parameters at the time the script is executed. Fortunately, Pig provides this capability via parameter substitution. There are four different mechanisms to define parameters that can be referenced in a Pig Latin script:

- Parameters can be defined as command line arguments; each parameter is passed to Pig as a separate argument using -param switches at script execution time
- Parameters can be defined in a parameter file that's passed to Pig using the -param_file command line argument when the script is executed
- Parameters can be defined inside Pig Latin scripts using the "%declare" and "%default" preprocessor statements

You can use none, one or any combination of the above options.

Let's look at an example Pig script that could be run to perform IIS log ingestion. The script loads and filters an IIS log looking for requests that didn't complete with status-code of 200 or 201.

Note that parameter names in Pig Latin scripts are preceded by a dollar sign, $. For example, the LOAD statement references six parameters; $WASB_SCHEME, $ROOT_FOLDER, $YEAR, $MONTH, $DAY and $INPUTFILE.

Note also the script makes use of the %default preprocessor statement to define default values for the WASB_SCHEME and ROOT_FOLDER parameters:

```
-- The "%default" preprocessor statement can be used to define a parameter

-- and specify its default value
```

```
--

%default WASB_SCHEME 'wasb';

%default ROOT_FOLDER 'logs'

-- Note how the below "LOAD" statement references the WASB_SCHEME and
ROOT_FOLDER

-- parameters that were defined with the default statement directly above

-- The values for the remaining four parameters; YEAR, MONTH, DAY and INPUTFILE
will be passed

-- from the command line using a parameter file or individual command line
arguments

--

iislog = LOAD
'$WASB_SCHEME:///$ROOT_FOLDER/$YEAR/$MONTH/$DAY/$INPUTFILE' USING
PigStorage(' ')

AS (date: chararray

, time: chararray

, sourceIP: chararray

, csMethod: chararray

, csUriStem: chararray

, csUriQuery: chararray

, sourcePort: chararray

, csUsername: chararray

, cIP: chararray

, csUserAgent: chararray

, csReferer: chararray

, scStatus: chararray

, scSubStatus: chararray

, scWin32Status: chararray

, timeTaken: chararray);

errors = FILTER iislog BY date == '$YEAR-$MONTH-$DAY' AND scStatus != '200' AND
scStatus != '201';

STORE errors INTO
'$WASB_SCHEME:///$ROOT_FOLDER/$YEAR/$MONTH/$DAY/output';
```