Pig Latin – Data Model

As discussed in the previous chapters, the data model of Pig is fully nested. A **Relation** is the outermost structure of the Pig Latin data model. And it is a **bag** where −

- A bag is a collection of tuples.
- A tuple is an ordered set of fields.
- A field is a piece of data.

Pig Latin – Statemets

While processing data using Pig Latin, **statements** are the basic constructs.

- These statements work with **relations**. They include **expressions** and **schemas**.
- Every statement ends with a semicolon (;).
- We will perform various operations using operators provided by Pig Latin, through statements.
- Except LOAD and STORE, while performing all other operations, Pig Latin statements take a relation as input and produce another relation as output.
- As soon as you enter a **Load** statement in the Grunt shell, its semantic checking will be carried out. To see the contents of the schema, you need to use the **Dump** operator. Only after performing the **dump** operation, the MapReduce job for loading the data into the file system will be carried out.

Example

Given below is a Pig Latin statement, which loads data to Apache Pig.

```
grunt> Student_data = LOAD 'student_data.txt' USING
PigStorage(',')as
   ( id:int, firstname:chararray, lastname:chararray,
phone:chararray, city:chararray );
```

Pig Latin – 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]

Null Values

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 – Arithmetic Operators

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 Latin – Comparison Operators

The following table describes the comparison operators of Pig Latin.

Operator	Description	Example
==	**Equal** − Checks if the values of two operands are equal or not; if yes, then the condition becomes true.	(a = b) is not true
!=	**Not Equal** − Checks if the values of two operands are equal or not. If the values are not equal, then condition becomes true.	(a != b) is true.
>	**Greater than** − Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition becomes true.	(a > b) is not true.
<	**Less than** − Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.	(a < b) is true.
>=	**Greater than or equal to** − Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition becomes true.	(a >= b) is not true.
<=	**Less than or equal to** − Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true.	(a <= b) is true.
matches	**Pattern matching** − Checks whether the string in the left-hand side matches with the constant in the right-hand side.	f1 matches '.*tutorial.*'

Pig Latin – Type Construction Operators

The following table describes the Type construction operators of Pig Latin.

Operator	Description	Example
()	**Tuple constructor operator** − This operator is used to construct a tuple.	(Raju, 30)
{}	**Bag constructor operator** − This operator is used to construct a bag.	{(Raju, 30), (Mohammad, 45)}
[]	**Map constructor operator** − This operator is used to construct a tuple.	[name#Raja, age#30]

Pig Latin – Relational Operations

The following table describes the relational operators of Pig Latin.

Operator	Description
Loading and Storing	
LOAD	To Load the data from the file system (local/HDFS) into a relation.
STORE	To save a relation to the file system (local/HDFS).
Filtering	
FILTER	To remove unwanted rows from a relation.
DISTINCT	To remove duplicate rows from a relation.
FOREACH, GENERATE	To generate data transformations based on columns of data.

STREAM	To transform a relation using an external program.
Grouping and Joining	
JOIN	To join two or more relations.
COGROUP	To group the data in two or more relations.
GROUP	To group the data in a single relation.
CROSS	To create the cross product of two or more relations.
Sorting	
ORDER	To arrange a relation in a sorted order based on one or more fields (ascending or descending).
LIMIT	To get a limited number of tuples from a relation.
Combining and Splitting	
UNION	To combine two or more relations into a single relation.
SPLIT	To split a single relation into two or more relations.
Diagnostic Operators	
DUMP	To print the contents of a relation on the console.
DESCRIBE	To describe the schema of a relation.
EXPLAIN	To view the logical, physical, or MapReduce execution plans to compute a relation.

ILLUSTRATE	To view the step-by-step execution of a series of statements.