Apache Pig:

Pig is a high-level programming language useful for analyzing large data sets. A pig was a result of development effort at Yahoo!.

In a MapReduce framework, programs need to be translated into a series of Map and Reduce stages. However, this is not a programming model which data analysts are familiar with. So, in order to bridge this gap, an abstraction called Pig was built on top of Hadoop.

Apache Pig enables people to focus more on **analyzing bulk data sets and to spend less time writing Map-Reduce programs.**

The high-level procedural language used in Apache Pig platform is called *Pig Latin*. Apache Pig features 'Pig Latin' which is a relatively simpler language which can run over distributed datasets on Hadoop File System (HDFS).

In Apache Pig, you need to write Pig scripts using Pig Latin language, which gets converted to MapReduce job when your run you Pig script.

Apache Pig has various operators which are used to perform the tasks like reading, writing , processing the data.

Installation of Apache Pig:

Step 1: Download Pig tar file.

Command: wget http://www-us.apache.org/dist/pig/pig-0.16.0/pig-0.16.0.tar.gz

Step 2: Extract the **tar** file using tar command. In below tar command, **x** means extract an archive file, **z**means filter an archive through gzip, **f** means filename of an archive file.

Command: tar -xzf pig-0.16.0.tar.gz

Step 3: Edit the "**.bashrc**" file to update the environment variables of Apache Pig. We are setting it so that we can access pig from any directory, we need not go to pig directory to execute pig commands. Also, if any other application is looking for Pig, it will get to know the path of Apache Pig from this file.

Command: gedit ~/.bashrc

Add the following at the end of the file:

```
# Set PIG_HOME

export PIG_HOME=/home/mamoon/pig-0.16.0
export PATH=$PATH:/home/mamoon/pig-0.16.0/bin
export PIG_CLASSPATH=$HADOOP_CONF_DIR
```

Run below command to make the changes get updated in same terminal.

Command: source ~/.bashrc

Step 4: Check pig version. This is to test that Apache Pig got installed correctly. In case, you don't get the Apache Pig version, you need to verify if you have followed the above steps correctly.

Command: pig –version

Step 5: Run Pig to start the grunt shell. Grunt shell is used to run Pig Latin scripts.
Command: pig

Execution modes in Apache Pig:

MapReduce Mode – This is the default mode, which requires access to a Hadoop cluster and HDFS installation. Since, this is a default mode, it is not necessary to specify -x flag (you can execute pig OR pig -x mapreduce). The input and output in this mode are present on HDFS.

Local Mode – With access to a single machine, all files are installed and run using a local host and file system. Here the local mode is specified using '-x flag' (pig -x local). The input and output in this mode are present on local file system.

Command: pig -x local

Apache Pig vs MapReduce:

- Programmers face difficulty writing MapReduce tasks as it requires Java or Python programming knowledge. For them, Apache Pig is a savior.

- Pig Latin is a high-level data flow language, whereas MapReduce is a low-level data processing paradigm. Without writing complex Java implementations in MapReduce, programmers can achieve the same implementations very easily using Pig Latin.

- Apache Pig uses multi-query approach (i.e. using a single query of Pig Latin we can accomplish multiple MapReduce tasks), which reduces the length of the code by 20 times. Hence, this reduces the development period by almost 16 times.

- Pig provides many built-in operators to support data operations like joins, filters, ordering, sorting etc. Whereas to perform the same function in MapReduce is a humongous task.

- Performing a Join operation in Apache Pig is simple. Whereas it is difficult in MapReduce to perform a Join operation between the data sets, as it requires multiple MapReduce tasks to be executed sequentially to fulfill the job. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce. I will explain you these data types in a while.

Architecture of Apache Pig:

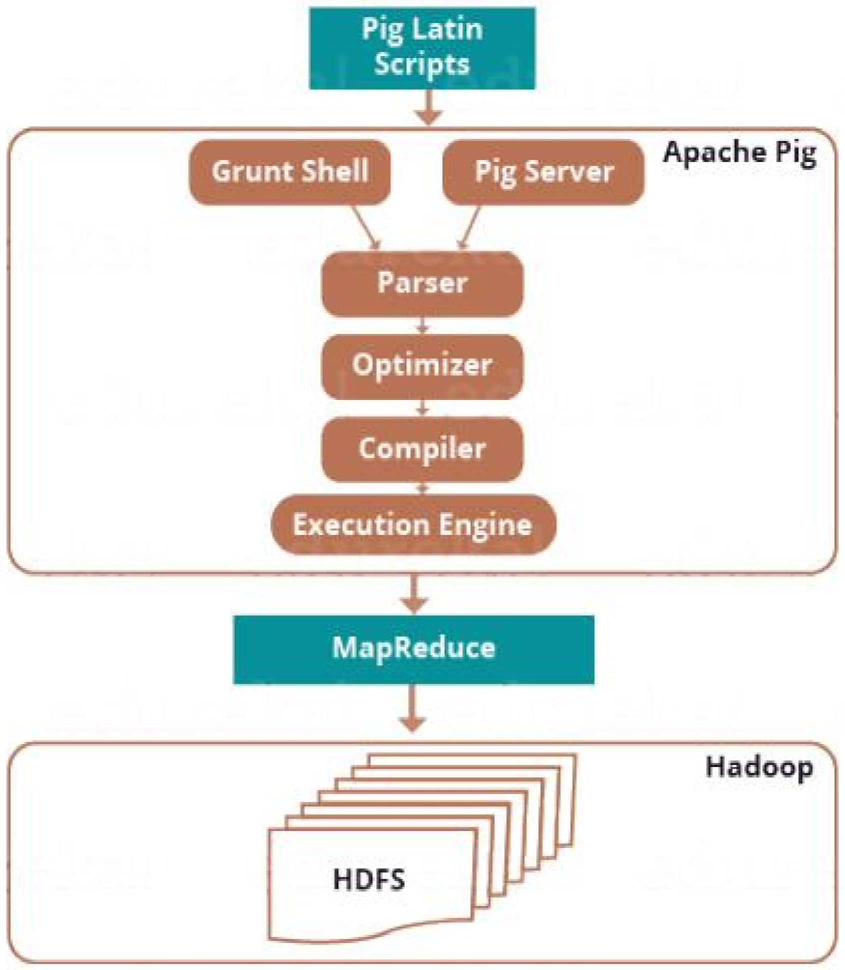


Figure: Apache Pig Architecture

Pig Latin Scripts:

Initially we submit Pig scripts to the Apache Pig execution environment which can be written in Pig Latin using built-in operators.

There are three ways to execute the Pig script:

- Grunt Shell: This is Pig's interactive shell provided to execute all Pig Scripts.

- Script File: Write all the Pig commands in a script file and execute the Pig script file. This is executed by the Pig Server.

- Embedded Script: If some functions are unavailable in built-in operators, we can programmatically create User Defined Functions to bring that functionalities using other languages like Java, Python, Ruby, etc. and embed it in Pig Latin Script file. Then, execute that script file.

Parser:

- After passing through Grunt or Pig Server, Pig Scripts are passed to the Parser.
- The Parser does type checking and checks the syntax of the script.
- The parser outputs a DAG (directed acyclic graph). DAG represents the Pig Latin statements and logical operators.
- The logical operators are represented as the nodes and the data flows are represented as edges.

Optimizer

- Then the DAG is submitted to the optimizer.
- The Optimizer performs the optimization activities like split, merge, transform, and reorder operators etc.
- This optimizer provides the automatic optimization feature to Apache Pig.
- The optimizer basically aims to reduce the amount of data in the pipeline at any instance of time while processing the extracted data, and for that it performs functions like:

- *PushUpFilter*: If there are multiple conditions in the filter and the filter can be split, Pig splits the conditions and pushes up each condition separately. Selecting these conditions earlier, helps in reducing the number of records remaining in the pipeline.

- *PushDownForEachFlatten*: Applying flatten, which produces a cross product between a complex type such as a tuple or a bag and the other fields in the record, as late as possible in the plan. This keeps the number of records low in the pipeline.

- ColumnPruner: Omitting columns that are never used or no longer needed, reducing the size of the record. This can be applied after each operator, so that fields can be pruned as aggressively as possible.

- MapKeyPruner: Omitting map keys that are never used, reducing the size of the record.

- LimitOptimizer: If the limit operator is immediately applied after a load or sort operator, Pig converts the load or sort operator into a limit-sensitive implementation, which does not require processing the whole data set. Applying the limit earlier, reduces the number of records.

Compiler

- After the optimization process, the compiler compiles the optimized code into a series of MapReduce jobs.

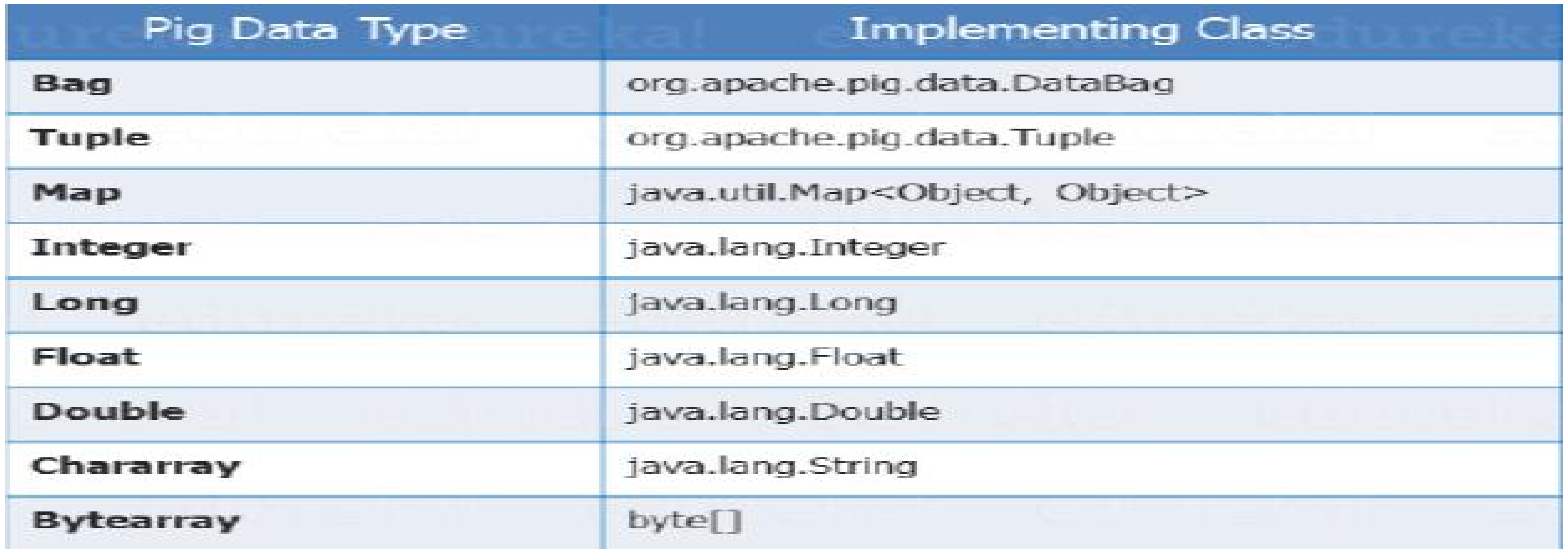
- The compiler is the one who is responsible for converting Pig jobs automatically into MapReduce jobs.

Execution Engine:

- Finally these MapReduce jobs are submitted for execution to the execution engine.
- Then the MapReduce jobs are executed and gives the required result.
- The result can be displayed on the screen using “DUMP” statement and can be stored in the HDFS using “STORE” statement.

Pig Data Types:

- The data model of Pig Latin enables Pig to handle all types of data.
- Pig Latin can handle both atomic data types like int, float, long, double etc. and complex data types like tuple, bag and map.



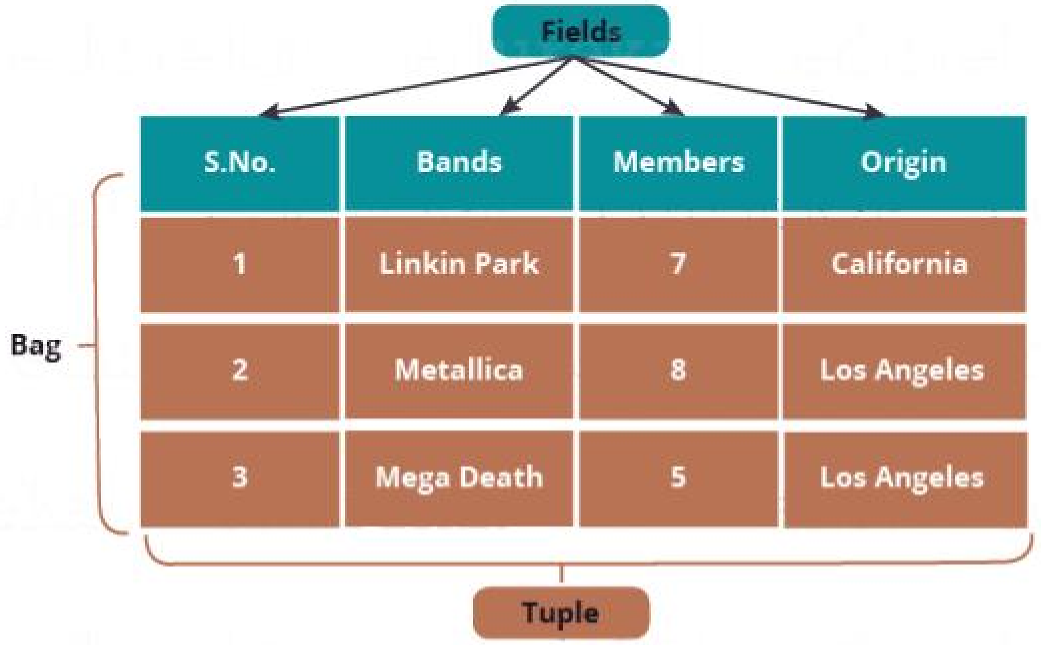
Pig Data Type	Implementing Class
Bag	org.apache.pig.data.DataBag
Tuple	org.apache.pig.data.Tuple
Map	java.util.Map<Object, Object>
Integer	java.lang.Integer
Long	java.lang.Long
Float	java.lang.Float
Double	java.lang.Double
Chararray	java.lang.String
Bytearray	byte[]

Atomic /Scalar Data Type:

Atomic or scalar data types are the basic data types which are used in all the languages like string, int, float, long, double, char[], byte[]. These are also called the primitive data types. The value of each cell in a field (column) is an atomic data type.

- For fields, positional indexes are generated by the system automatically (also known as positional notation), which is represented by '$' and it starts from $0, and grows $1, $2, so on… As compared with the below image $0 = S.No., $1 = Bands, $2 = Members, $3 = Origin.

- Scalar data types are – '1', 'Linkin Park', '7', 'California' etc.



S.No.	Bands	Members	Origin
1	Linkin Park	7	California
2	Metallica	8	Los Angeles
3	Mega Death	5	Los Angeles

Figure: Apache Pig Data Model

Tuple:

•Tuple is an ordered set of fields which may contain different data types for each field.
•A Tuple is a set of cells from a single row as shown in the above image.
•The elements inside a tuple does not necessarily need to have a schema attached to it.
•A tuple is represented by ‘()’ symbol.
Example of tuple − (1, Linkin Park, 7, California)
•Since tuples are ordered, we can access fields in each tuple using indexes of the fields, like $1 from above tuple will return a value ‘Linkin Park’.

Bag:

•A bag is a collection of a set of tuples and these tuples are subset of rows or entire rows of a table.
•A bag can contain duplicate tuples, and it is not mandatory that they need to be unique.
•The bag has a flexible schema i.e. tuples within the bag can have different number of fields. A bag can also have tuples with different data types.
•A bag is represented by ‘{}’ symbol.
Example of a bag − **{(Linkin Park, 7, California), (Metallica, 8), (Mega Death, Los Angeles)}**
•But for Apache Pig to effectively process bags, the fields and their respective data types need to be in the same sequence.

Set of bags –

{(Linkin Park, 7, California), (Metallica, 8), (Mega Death, Los Angeles)},

{(Metallica, 8, Los Angeles), (Mega Death, 8), (Linkin Park, California)}

•There are two types of Bag, i.e. Outer Bag or relations and Inner Bag.

•Outer bag or relation is noting but a bag of tuples. Here relations are similar as relations in relational databases. To understand it better let us take an example:

{(Linkin Park, California), (Metallica, Los Angeles), (Mega Death, Los Angeles)}

This above bag explains the relation between the *Band* and their place of *Origin*.

•On the other hand, an inner bag contains a bag inside a tuple. For Example, if we sort *Band* tuples based on *Band's Origin*, we will get:

(Los Angeles, **{(Metallica, Los Angeles), (Mega Death, Los Angeles)}**)
(California,**{(Linkin Park, California)}**)

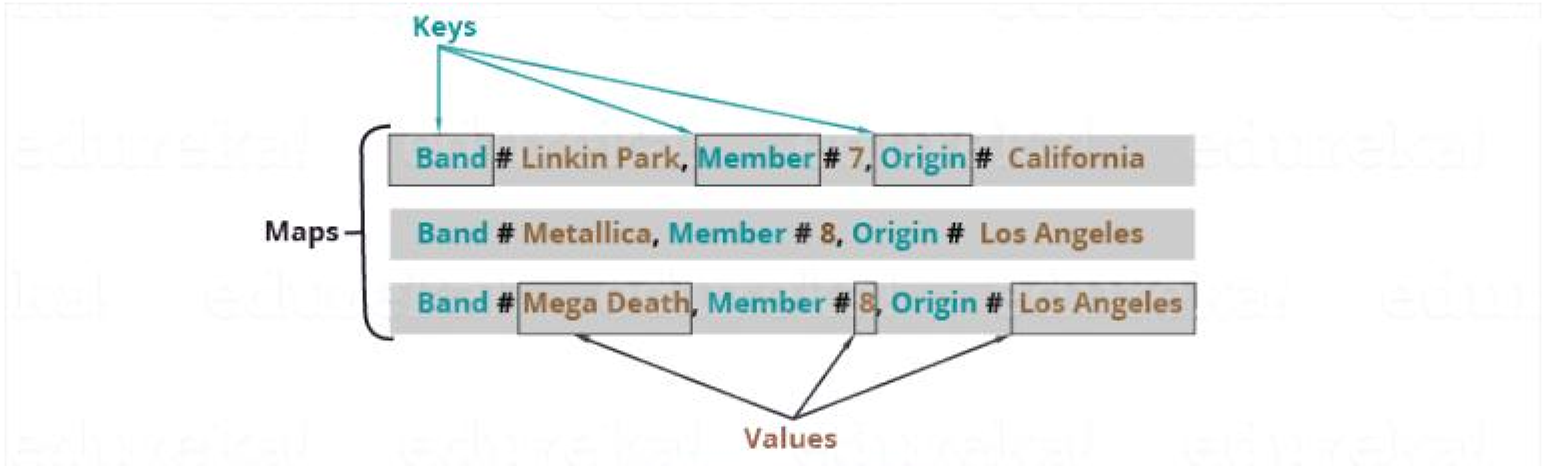
•First field type is a string while the second field type is a bag, which is an inner bag within a tuple.

Map:

•A map is key-value pairs used to represent data elements.
•The key must be a chararray [] and should be unique like column name, so it can be indexed and value associated with it can be accessed on basis of the keys.
•The value can be of any data type.

Maps are represented by '[]' symbol and key-value are separated by '#' symbol, as you can see in the above image.

Example of maps− [band#Linkin Park, members#7], [band#Metallica, members#8]


Keys
Maps
Band # Linkin Park, Member # 7, Origin # California
Band # Metallica, Member # 8, Origin # Los Angeles
Band # Mega Death, Member # 8, Origin # Los Angeles
Values
Figure: Map Example

Reading Data in Apache Pig:

Step1: Prepare HDFS File Storage.

In MapReduce mode, Pig reads (loads) data from HDFS and stores the results back in HDFS. Therefore, let us start HDFS and load employ.txt file from local system to HDFS.

Step2: Create Directory in HDFS

```
$ hdfs dfs –mkdir /home/PigData
```

Step3: Place the data in HDFS Directory

```
$hadoop fs –copyFromLocal /home/mamoon/employ.txt   /home/PigData/
```

Step4: Verifying Loaded File employ.txt on HDFS

```
$hdfs dfs -cat /home/PigData/employ.txt
```

Step5: Load data into Apache Pig from the file system (HDFS/ Local) using **LOAD** operator of **Pig Latin**.

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.

Syntax of 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);
```

Step6: Start the Pig Grunt shell

```
$ pig
```

Step7: Execute the Load Statement

```
grunt> employee= LOAD ‘/home/PigData/employ.txt’
   USING PigStorage(‘,’)
   as (id:int, name:chararray, age:int, address:chararray, salary:int, department:chararray);
```

Storing Data in Apache Pig:

We can store the loaded data in the file system using the **store** operator.

Syntax of store operator:

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

Store the relation in the HDFS directory **“/PigOut/”** as shown below.

```
grunt> STORE employee INTO ' /home/PigData/PigOut/' USING PigStorage (',');
```

Apache Pig - Diagnostic Operators:

The load statement will simply load the data into the specified relation in Apache Pig. To verify the execution of the Load statement, you have to use the Diagnostic Operators. Pig Latin provides four different types of diagnostic operators −

•Dump operator
•Describe operator
•Explanation operator
•Illustration operator

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

```
grunt> Dump Relation_Name
```

Describe Operator:

The describe operator is used to view the schema of a relation.

Syntax
The syntax of the describe operator is as follows −

```
grunt> Describe Relation_name
```

Explain Operator:

The explain operator is used to display the logical, physical, and MapReduce execution plans of a relation.

Syntax

```
grunt> explain Relation_name;
```

Illustrate Operator:

The illustrate operator gives you the step-by-step execution of a sequence of statements.

Syntax

```
grunt> illustrate Relation_name;
```

Grouping and Joining in Pig:

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

Syntax

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

Example:

1. Assume that we have a file named employ.txt in the HDFS directory /PigData/

2. We have loaded this file into Apache Pig with the relation name employee as shown below.

```
    grunt> employee= LOAD ‘/home/PigData/employ.txt’
    USING PigStorage(‘,’)
    as (id:int, name:chararray, age:int, address:chararray, salary:int, department:chararray);
```

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

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

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

```
grunt> Dump group_data;
```

5. You will get output displaying the contents of the relation named group_data as shown below. Here you can observe that the resulting schema has two columns −

•One is age, by which we have grouped the relation.

•The other is a bag, which contains the group of tuples, student records with the respective age.

6. You can see the schema of the table after grouping the data using the describe command as shown below.

```
grunt> Describe group_data;
```

7. You can get the sample illustration of the schema using the **illustrate** command as shown below.

```
grunt> Illustrate group_data;
```

Grouping by Multiple Columns:

Let us group the relation by age and city as shown below.

```
grunt> group_multiple = GROUP employee by (age, department);
```

You can verify the content of the relation named group_multiple using the Dump operator as shown below.

```
grunt> Dump group_multiple;
```

Group All:

You can group a relation by all the columns as shown below.

```
grunt> group_all = GROUP employee All;
```

Now, verify the content of the relation group_all as shown below.

```
grunt> Dump group_all;
```

Join Operator:

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

Example:
1. Assume that we have two files namely customers.txt and orders.txt in the /PigData/ 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
```

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

```
grunt> customers = LOAD ‘home/PigData/customers.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);

grunt> orders = LOAD 'home/PigData/orders.txt' USING PigStorage(',')
   as (oid:int, date:chararray, customer_id:int, amount:int);
```

3. Let us now perform various Join operations on these two relations.

Self – join

Self-join is used to join a table with itself as if the table were two relations, temporarily renaming at least one relation.

Generally, in Apache Pig, to perform self-join, we will load the same data multiple times, under different aliases (names). Therefore let us load the contents of the file customers.txt as two tables as shown below.

```
grunt> customers1 = LOAD ‘home/PigData/customers.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);

grunt> customers2 = LOAD 'home/PigData/customers.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);
```

Syntax

Given below is the syntax of performing self-join operation using the JOIN operator.

```
grunt> Relation3_name = JOIN Relation1_name BY key, Relation2_name BY key ;
```

Example

1. Perform self-join operation on the relation customers, by joining the two relations customers1 and customers2 as shown below.

```
grunt> customers3 = JOIN customers1 BY id, customers2 BY id;
```

2. Verify the relation customers3 using the DUMP operator as shown below.

```
grunt> Dump customers3;
```

3. It will produce the following output, displaying the contents of the relation customers.

```
(1,Ramesh,32,Ahmedabad,2000,1,Ramesh,32,Ahmedabad,2000)
(2,Khilan,25,Delhi,1500,2,Khilan,25,Delhi,1500)
(3,kaushik,23,Kota,2000,3,kaushik,23,Kota,2000)
(4,Chaitali,25,Mumbai,6500,4,Chaitali,25,Mumbai,6500)
(5,Hardik,27,Bhopal,8500,5,Hardik,27,Bhopal,8500)
(6,Komal,22,MP,4500,6,Komal,22,MP,4500)
(7,Muffy,24,Indore,10000,7,Muffy,24,Indore,10000)
```

Inner Join

Inner Join is used quite frequently; it is also referred to as equijoin. An inner join returns rows when there is a match in both tables.

It creates a new relation by combining column values of two relations (say A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, the column values for each matched pair of rows of A and B are combined into a result row.

Syntax
Here is the syntax of performing inner join operation using the JOIN operator.

```
grunt> result = JOIN relation1 BY columnname, relation2 BY columnname;
```

Example

1. Perform inner join operation on the two relations customers and orders as shown below.

```
grunt> coustomer_orders = JOIN customers BY id, orders BY customer_id;
```

2. Verify the relation coustomer_orders using the DUMP operator as shown below.

```
grunt> Dump coustomer_orders;
```

3. You will get the following output that will the contents of the relation named coustomer_orders.

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
```

Outer Join:

Unlike inner join, outer join returns all the rows from at least one of the relations. An outer join operation is carried out in three ways −

•Left outer join
•Right outer join
•Full outer join

Left Outer Join:

The left outer Join operation returns all rows from the left table, even if there are no matches in the right relation.

Syntax
Given below is the syntax of performing left outer join operation using the JOIN operator.

```
grunt> Relation3_name = JOIN Relation1_name BY id LEFT OUTER, Relation2_name BY customer_id;
```

Example
1. Perform left outer join operation on the two relations customers and orders as shown below.
   ```
   grunt> outer_left = JOIN customers BY id LEFT OUTER, orders BY customer_id;
   ```
2. Verify the relation outer_left using the DUMP operator as shown below.

```
grunt> Dump outer_left;
```

3. It will produce the following output, displaying the contents of the relation outer_left.

```
(1,Ramesh,32,Ahmedabad,2000,,,,)
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
(5,Hardik,27,Bhopal,8500,,,,)
(6,Komal,22,MP,4500,,,,)
(7,Muffy,24,Indore,10000,,,,)
```

Right Outer Join:

The right outer join operation returns all rows from the right table, even if there are no matches in the left table.

Syntax
Given below is the syntax of performing right outer join operation using the JOIN operator.

```
grunt> outer_right = JOIN customers BY id RIGHT, orders BY customer_id;
```

Example

1. Perform right outer join operation on the two relations customers and orders as shown below.

```
grunt> outer_right = JOIN customers BY id RIGHT, orders BY customer_id;
```

2. Verify the relation outer_right using the DUMP operator as shown below.

```
grunt> Dump outer_right
```

3. It will produce the following output, displaying the contents of the relation outer_right.

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
```

Full Outer Join:

The full outer join operation returns rows when there is a match in one of the relations.

Syntax
Given below is the syntax of performing full outer join using the JOIN operator.

```
grunt> outer_full = JOIN customers BY id FULL OUTER, orders BY customer_id;
```

Example
1. Perform full outer join operation on the two relations customers and orders as shown below.

```
grunt> outer_full = JOIN customers BY id FULL OUTER, orders BY customer_id;
```

2. Verify the relation outer_full using the DUMP operator as shown below.

```
grunt> Dump outer_full;
```

3. It will produce the following output, displaying the contents of the relation outer_full.

```
(1,Ramesh,32,Ahmedabad,2000,,,,)
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
(5,Hardik,27,Bhopal,8500,,,,)
(6,Komal,22,MP,4500,,,,)
(7,Muffy,24,Indore,10000,,,,)
```

Combining and Splitting in Pig:

The UNION operator of Pig Latin is used to merge the content of two relations. To perform UNION operation on two relations, their columns and domains must be identical.

Syntax

```
grunt> Relation_name3 = UNION Relation_name1, Relation_name2;
```

Example

1. Assume that we have two files namely student_data1.txt and student_data2.txt in the /PigData/ directory of HDFS as shown below.

Student_data1.txt

```
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.
```

Student_data2.txt

```
7,Komal,Nayak,9848022334,trivendram.
8,Bharathi,Nambiayar,9848022333,Chennai.
```

2. We have loaded these two files into Pig with the relations student1 and student2 as shown below.

```
grunt> student1 = LOAD ‘home/PigData/student_data1.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);

grunt> student2 = LOAD 'home/PigData/student_data2.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
```

3. Now merge the contents of these two relations using the UNION operator as shown below.

```
grunt> student = UNION student1, student2;
```

4. Verify the relation student using the DUMP operator as shown below.

```
grunt> Dump student;
```

5. It will display the following output, displaying the contents of the relation student.

```
(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)
(7,Komal,Nayak,9848022334,trivendram)
(8,Bharathi,Nambiayar,9848022333,Chennai)
```

Split Operator:

The SPLIT operator is used to split a relation into two or more relations.

```
grunt> SPLIT Relation1_name INTO Relation2_name IF (condition1), Relation2_name (condition2),
```

Example
1. Assume that we have a file named student_details.txt in the HDFS directory /PigData/ 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
```

2. We have loaded this file into Pig with the relation name student_details as shown below.

```
student_details = LOAD ‘home/PigData/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
```

3. Split the relation into two, one listing the employees of age less than 23, and the other listing the employees having the age between 22 and 25.

```
grunt> SPLIT student_details into student_details1 if age<23, student_details2 if (22<age and age>25);
```

4. Verify the relations student_details1 and student_details2 using the DUMP operator as shown below.

```
grunt> Dump student_details1;
grunt> Dump student_details2;
```

5. It will produce the following output, displaying the contents of the relations student_details1 and student_details2 respectively.

```
grunt> Dump student_details1;
(1,Rajiv,Reddy,21,9848022337,Hyderabad)
(2,siddarth,Battacharya,22,9848022338,Kolkata)
(3,Rajesh,Khanna,22,9848022339,Delhi)
(4,Preethi,Agarwal,21,9848022330,Pune)

grunt> Dump student_details2;
(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)
(6,Archana,Mishra,23,9848022335,Chennai)
(7,Komal,Nayak,24,9848022334,trivendram)
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
```

Pig Filtering:

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

Syntax

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

Example:

1. Assume that we have a file named student_details.txt in the HDFS directory /PigData/ 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
```

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

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

3. 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';
```

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

```
grunt> Dump filter_data;
```

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

```
(6,Archana,Mishra,23,9848022335,Chennai)
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
```

DISTINCT Operator:

The DISTINCT operator is used to remove redundant (duplicate) tuples from a relation.

Syntax

```
grunt> Relation_name2 = DISTINCT Relatin_name1;
```

Example

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

student_details.txt

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

2. We have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'home/PigData/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
```

3. Now remove the redundant (duplicate) tuples from the relation named student_details using the DISTINCT operator, and store it as another relation named distinct_data as shown below.

```
grunt> distinct_data = DISTINCT student_details;
```

4. Verify the relation distinct_data using the DUMP operator as shown below.

```
grunt> Dump distinct_data;
```

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

```
(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)
```

FOREACH Operator :

The FOREACH operator is used to generate specified data transformations based on the column data.

Syntax

```
grunt> Relation_name2 = FOREACH Relation_name1 GENERATE (required data);
```

Example

1. Assume that we have a file named student_details.txt in the HDFS directory /PigData/ 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
```

2. We have loaded this file into Pig with the relation name student_details as shown below.

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

3. Now get the id, age, and city values of each student from the relation student_details and store it into another relation named foreach_data using the foreach operator as shown below.

```
grunt> foreach_data = FOREACH student_details GENERATE id,age,city;
```

4. Verify the relation foreach_data using the DUMP operator as shown below.

```
grunt> Dump foreach_data;
```

5. It will produce the following output, displaying the contents of the relation foreach_data.

```
(1,21,Hyderabad)
(2,22,Kolkata)
(3,22,Delhi)
(4,21,Pune)
(5,23,Bhuwaneshwar)
(6,23,Chennai)
(7,24,trivendram)
(8,24,Chennai)
```

Pig Sorting:

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

Syntax

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

Example
1. 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
```

2. We have loaded this file into Pig with the relation name student_details as shown below.

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

3. 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;
```

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

```
grunt> Dump order_by_data;
```

5. It will produce the following output, displaying the contents of the relation order_by_data.

```
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
(7,Komal,Nayak,24,9848022334,trivendram)
(6,Archana,Mishra,23,9848022335,Chennai)
(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)
(3,Rajesh,Khanna,22,9848022339,Delhi)
(2,siddarth,Battacharya,22,9848022338,Kolkata)
(4,Preethi,Agarwal,21,9848022330,Pune)
(1,Rajiv,Reddy,21,9848022337,Hyderabad)
```

LIMIT Operator:

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

Syntax

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

Example

1. Assume that we have a file named student_details.txt in the HDFS directory /PigData/ 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
```

2. 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);
```

3. 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;
```

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

```
grunt> Dump limit_data;
```

5. 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)
```

Running Pig Scripts 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.

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

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

Step 3
Execute it from the Grunt shell as well using the exec command as shown below.

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

Step 4
Executing a Pig Script from HDFS

Execute a Pig script that resides in the HDFS. Suppose there is a Pig script with the name Sample_script.pig in the HDFS directory named /PigData/. We can execute it as shown below.

```
$ pig -x mapreduce home/PigData/Sample_script.pig
```

Example
1. Assume we have a file student_details.txt in HDFS with the following content.

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
```

2. Put sample script with the name sample.pig, in the same HDFS directory. This file contains statements performing operations and transformations on the student relation, as shown below.

```
student = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
student_order = ORDER student BY age DESC;
student_limit = LIMIT student_order 4;
Dump student_limit;
```

3. Execute the sample.pig as shown below.

```
$ pig -x mapreduce home/PigData/sample.pig
```

4. Apache Pig gets executed and gives you the output with the following content.

```
(7,Komal,Nayak,24,9848022334,trivendram)
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)
(6,Archana,Mishra,23,9848022335,Chennai)
2015-10-19 10:31:27,446 [main] INFO  org.apache.pig.Main - Pig script completed in 12
minutes, 32 seconds and 751 milliseconds (752751 ms)
```

Apache Pig - Eval Functions

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

6	DIFF()To compare two bags (fields) in a tuple.
7	IsEmpty()To check if a bag or map is empty.
8	MAX()To calculate the highest value for a column (numeric values or chararrays) in a single-column bag.
9	MIN()To get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag.
10	PluckTuple()Using the Pig Latin **PluckTuple()** function, we can define a string Prefix and filter the columns in a relation that begin with the given prefix.
11	SIZE()To compute the number of elements based on any Pig data type.
12	SUBTRACT()To subtract two bags. It takes two bags as inputs and returns a bag which contains the tuples of the first bag that are not in the second bag.
13	SUM()To get the total of the numeric values of a column in a single-column bag.
14	TOKENIZE()To split a string (which contains a group of words) in a single tuple and return a bag which contains the output of the split operation.

Apache Pig - AVG()

The Pig-Latin AVG() function is used to compute the average of the numerical values within a bag. While calculating the average value, the AVG() function ignores the NULL values.

Note −

To get the global average value, we need to perform a Group All operation, and calculate the average value using the AVG() function.

To get the average value of a group, we need to group it using the Group By operator and proceed with the average function.

Syntax
Given below is the syntax of the AVG() function.

```
grunt> AVG(expression)
```

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,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

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, gpa:int);
```

Calculating the Average GPA

We can use the built-in function AVG() (case-sensitive) to calculate the average of a set of numerical values. Let’s group the relation student_details using the Group All operator, and store the result in the relation named student_group_all as shown below.

```
grunt> student_group_all = Group student_details All;
```

This will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate the global average GPA of all the students using the AVG() function as shown below.

```
grunt> student_gpa_avg = foreach student_group_all  Generate
   (student_details.firstname, student_details.gpa), AVG(student_details.gpa);
```

Verification
Verify the relation student_gpa_avg using the DUMP operator as shown below.

```
grunt> Dump student_gpa_avg;
```

Output
It will display the contents of the relation student_gpa_avg as follows.

```
(({(Bharathi),(Komal),(Archana),(Trupthi),(Preethi),(Rajesh),(siddarth),(Rajiv) },
  {   (72)   ,  (83) ,   (87)  ,   (75)  ,   (93)  ,  (90)  ,   (78)   ,  (89)  }),83.375)
```

Apache Pig - BagToString():

The Pig Latin BagToString() function is used to concatenate the elements of a bag into a string. While concatenating, we can place a delimiter between these values (optional).

Generally bags are disordered and can be arranged by using ORDER BY operator.

Syntax

```
grunt> BagToString(vals:bag [, delimiter:chararray])
```

Example
Assume that we have a file named dateofbirth.txt in the HDFS directory /pig_data/ as shown below. This file contains the date-of-births.

dateofbirth.txt

```
22,3,1990
23,11,1989
1,3,1998
2,6,1980
26,9,1989
```

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

```
grunt> dob = LOAD 'hdfs://localhost:9000/pig_data/dateofbirth.txt' USING PigStorage(',')
   as (day:int, month:int, year:int);
```

Converting Bag to String:

Using the bagtostring() function, we can convert the data in the bag to string. Let us group the dob relation. The group operation will produce a bag containing all the tuples of the relation.

Group the relation dob using the Group All operator, and store the result in the relation named group_dob as shown below.

```
grunt> group_dob = Group dob All;
```

It will produce a relation as shown below.

```
grunt> Dump group_dob;

(all,{(26,9,1989),(2,6,1980),(1,3,1998),(23,11,1989),(22,3,1990)})
```

Here, we can observe a bag having all the date-of-births as tuples of it. Now, let’s convert the bag to string using the function BagToString().

```
grunt> dob_string = foreach group_dob Generate BagToString(dob);
```

Verification

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

```
grunt> Dump dob_string;
```

Output

It will produce the following output, displaying the contents of the relation dob_string.

```
(26_9_1989_2_6_1980_1_3_1998_23_11_1989_22_3_1990)
```

The CONCAT() function of Pig Latin is used to concatenate two or more expressions of the same type.

Syntax

```
grunt> CONCAT (expression, expression, [...expression])
```

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,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

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:chara
```

Concatenate these two values using the CONCAT() function.

```
grunt> student_name_concat = foreach student_details Generate CONCAT (firstname, lastname);
```

Verification

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

```
grunt> Dump student_name_concat;
```

Output

It will produce the following output, displaying the contents of the relation student_name_concat.

```
(RajivReddy)
(siddarthBattacharya)
(RajeshKhanna)
(PreethiAgarwal)
(TrupthiMohanthy)
(ArchanaMishra)
(KomalNayak)
(BharathiNambiayar)
```

We can also use an optional delimiter between the two expressions as shown below.

```
grunt> CONCAT(firstname, '_',lastname);
```

Now, let us concatenate the first name and last name of the student records in the student_details relation by placing ‘_’ between them as shown below.

```
grunt> student_name_concat = foreach student_details GENERATE CONCAT(firstname, '_',lastname);
```

Verification

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

```
grunt> Dump student_name_concat;
```

Output

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

```
(Rajiv_Reddy)
(siddarth_Battacharya)
(Rajesh_Khanna)
(Preethi_Agarwal)
(Trupthi_Mohanthy)
(Archana_Mishra)
(Komal_Nayak)
(Bharathi_Nambiayar)
```

Apache Pig - COUNT()

The COUNT() function of Pig Latin is used to get the number of elements in a bag. While counting the number of tuples in a bag, the COUNT() function ignores (will not count) the tuples having a NULL value in the FIRST FIELD.

Note −

To get the global count value (total number of tuples in a bag), we need to perform a Group All operation, and calculate the count value using the COUNT() function.

To get the count value of a group (Number of tuples in a group), we need to group it using the Group By operator and proceed with the count function.

Syntax
Given below is the syntax of the COUNT() function.

```
grunt> COUNT(expression)
```

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,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation named 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, gpa:int);
```

Calculating the Number of Tuples

We can use the built-in function COUNT() (case sensitive) to calculate the number of tuples in a relation. Let us group the relation student_details using the Group All operator, and store the result in the relation named student_group_all as shown below.

```
grunt> student_group_all = Group student_details All;
```

It will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate number of tuples/records in the relation.

```
grunt> student_count = foreach student_group_all  Generate COUNT(student_details.gpa);
```

Verification

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

```
grunt> Dump student_count;
```

Output

It will produce the following output, displaying the contents of the relation student_count.

```
8
```

Apache Pig - COUNT_STAR()

The COUNT_STAR() function of Pig Latin is similar to the COUNT() function. It is used to get the number of elements in a bag. While counting the elements, the COUNT_STAR() function includes the NULL values.

Note −

To get the global count value (total number of tuples in a bag), we need to perform a Group All operation, and calculate the count_star value using the COUNT_STAR() function.

To get the count value of a group (Number of tuples in a group), we need to group it using the Group By operator and proceed with the count_star function.

Syntax
Given below is the syntax of the COUNT_STAR() function.

```
grunt> COUNT_STAR(expression)
```

Example
Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below. This file contains an empty record.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

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, gpa:int);
```

Calculating the Number of Tuples

We can use the built-in function COUNT_STAR() to calculate the number of tuples in a relation. Let us group the relation student_details using the Group All operator, and store the result in the relation named student_group_all as shown below.

```
grunt> student_group_all = Group student_details All;
```

It will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89),
( , , , , , , )})
```

Let us now calculate the number of tuples/records in the relation.

```
grunt> student_count = foreach student_group_all  Generate COUNT_STAR(student_details.gpa);
```

Verification
Verify the relation student_count using the DUMP operator as shown below.

```
grunt> Dump student_count;
```

Output
It will produce the following output, displaying the contents of the relation student_count.

```
9
```

Since we have used the function COUNT_STAR(), it included the null tuple and returned 9.

Apache Pig - DIFF()

The DIFF() function of Pig Latin is used to compare two bags (fields) in a tuple. It takes two fields of a tuple as input and matches them. If they match, it returns an empty bag. If they do not match, it finds the elements that exist in one field (bag) and not found in the other, and returns these elements by wrapping them within a bag.

Syntax

```
grunt> DIFF (expression, expression)
```

Example
Generally the DIFF() function compares two bags in a tuple. Given below is its example, here we create two relations, cogroup them, and calculate the difference between them.

Assume that we have two files namely emp_sales.txt and emp_bonus.txt in the HDFS directory /pig_data/ as shown below. The emp_sales.txt contains the details of the employees of the sales department and the emp_bonus.txt contains the employee details who got bonus.

```
emp_sales.txt
1,Robin,22,25000,sales
2,BOB,23,30000,sales
3,Maya,23,25000,sales
4,Sara,25,40000,sales
5,David,23,45000,sales
6,Maggy,22,35000,sales

emp_bonus.txt
1,Robin,22,25000,sales
2,Jaya,23,20000,admin
3,Maya,23,25000,sales
4,Alia,25,50000,admin
5,David,23,45000,sales
6,Omar,30,30000,admin
```

And we have loaded these files into Pig, with the relation names emp_sales and emp_bonus respectively.

```
grunt> emp_sales = LOAD ‘/home/pig_data/emp_sales.txt' USING PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int, dept:chararray);

grunt> emp_bonus = LOAD ‘/home/pig_data/emp_bonus.txt' USING PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int, dept:chararray);
```

Group the records/tuples of the relations emp_sales and emp_bonus with the key sno, using the COGROUP operator as shown below.

```
grunt> cogroup_data = COGROUP emp_sales by sno, emp_bonus by sno;
```

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

```
grunt> Dump cogroup_data;

(1,{(1,Robin,22,25000,sales)},{(1,Robin,22,25000,sales)})
(2,{(2,BOB,23,30000,sales)},{(2,Jaya,23,20000,admin)})
(3,{(3,Maya,23,25000,sales)},{(3,Maya,23,25000,sales)})
(4,{(4,Sara,25,40000,sales)},{(4,Alia,25,50000,admin)})
(5,{(5,David,23,45000,sales)},{(5,David,23,45000,sales)})
(6,{(6,Maggy,22,35000,sales)},{(6,Omar,30,30000,admin)})
```

Calculating the Difference between Two Relations

Let us now calculate the difference between the two relations using DIFF() function and store it in the relation diff_data as shown below.

```
grunt> diff_data = FOREACH cogroup_data GENERATE DIFF(emp_sales,emp_bonus);
```

Verification
Verify the relation diff_data using the DUMP operator as shown below.

```
grunt> Dump diff_data;

({})
({(2,BOB,23,30000,sales),(2,Jaya,23,20000,admin)})
({})
({(4,Sara,25,40000,sales),(4,Alia,25,50000,admin)})
({})
({(6,Maggy,22,35000,sales),(6,Omar,30,30000,admin)})
```

Apache Pig - IsEmpty()

The IsEmpty() function of Pig Latin is used to check if a bag or map is empty.

Syntax

```
grunt> IsEmpty(expression)
```

Example
Assume that we have two files namely emp_sales.txt and emp_bonus.txt in the HDFS directory /pig_data/ as shown below. The emp_sales.txt contains the details of the employees of the sales department and the emp_bonus.txt contains the employee details who got bonus.

emp_sales.txt

```
1,Robin,22,25000,sales
2,BOB,23,30000,sales
3,Maya,23,25000,sales
4,Sara,25,40000,sales
5,David,23,45000,sales
6,Maggy,22,35000,sales
```

emp_bonus.txt

```
1,Robin,22,25000,sales
2,Jaya,23,20000,admin
3,Maya,23,25000,sales
4,Alia,25,50000,admin
5,David,23,45000,sales
6,Omar,30,30000,admin
```

And we have loaded these files into Pig, with the relation names emp_sales and emp_bonus respectively, as shown below.

```
grunt> emp_sales = LOAD ‘/home/pig_data/emp_sales.txt' USING PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int, dept:chararray);

grunt> emp_bonus = LOAD ‘/home/pig_data/emp_bonus.txt' USING PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int, dept:chararray);
```

Let us now group the records/tuples of the relations emp_sales and emp_bonus with the key age, using the cogroup operator as shown below.

```
grunt> cogroup_data = COGROUP emp_sales by age, emp_bonus by age;
```

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

```
grunt> Dump cogroup_data;

(22,{(6,Maggy,22,35000,sales),(1,Robin,22,25000,sales)}, {(1,Robin,22,25000,sales)})
(23,{(5,David,23,45000,sales),(3,Maya,23,25000,sales),(2,BOB,23,30000,sales)},
   {(5,David,23,45000,sales),(3,Maya,23,25000,sales),(2,Jaya,23,20000,admin)})
(25,{(4,Sara,25,40000,sales)},{(4,Alia,25,50000,admin)})
(30,{},{(6,Omar,30,30000,admin)})
```

Getting the Groups having Empty Bags:

Let’s list such empty bags from the emp_sales relation in the group using the IsEmpty() function.

```
grunt> isempty_data = filter cogroup_data by IsEmpty(emp_sales);
```

Verification

Verify the relation isempty_data using the DUMP operator as shown below. The emp_sales relation holds the tuples that are not there in the relation emp_bonus.

```
grunt> Dump isempty_data;

(30,{},{(6,Omar,30,30000,admin)})
```

Apache Pig - MAX()

The Pig Latin MAX() function is used to calculate the highest value for a column (numeric values or chararrays) in a single-column bag. While calculating the maximum value, the Max() function ignores the NULL values.

Note −

To get the global maximum value, we need to perform a Group All operation, and calculate the maximum value using the MAX() function.

To get the maximum value of a group, we need to group it using the Group By operator and proceed with the maximum function.

Syntax

```
grunt> Max(expression)
```

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,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

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

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

Calculating the Maximum GPA

We can use the built-in function MAX() (case-sensitive) to calculate the maximum value from a set of given numerical values. Let us group the relation student_details using the Group All operator, and store the result in the relation named student_group_all as shown below.

```
grunt> student_group_all = Group student_details All;
```

This will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate the global maximum of GPA, i.e., maximum among the GPA values of all the students using the MAX() function as shown below.

```
grunt> student_gpa_max = foreach student_group_all  Generate
   (student_details.firstname, student_details.gpa), MAX(student_details.gpa);
```

Verification

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

```
grunt> Dump student_gpa_max;
```

Output

It will produce the following output, displaying the contents of the relation student_gpa_max.

```
(({(Bharathi),(Komal),(Archana),(Trupthi),(Preethi),(Rajesh),(siddarth),(Rajiv) } ,
    {    (72)    ,   (83)  ,     (87)   ,    (75)   ,   (93)   ,   (90)   ,     (78)   ,  (89)    }) ,93)
```

Apache Pig - MIN()

The MIN() function of Pig Latin is used to get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag. While calculating the minimum value, the MIN() function ignores the NULL values.

Note −

To get the global minimum value, we need to perform a Group All operation, and calculate the minimum value using the MIN() function.

To get the minimum value of a group, we need to group it using the Group By operator and proceed with the minimum function.

Syntax
Given below is the syntax of the MIN() function.

```
grunt> MIN(expression)
```

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,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

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

```
grunt> student_details = LOAD ‘/home/pig_data/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:charaarray, gpa:int);
```

Calculating the Minimum GPA

We can use the built-in function MIN() (case sensitive) to calculate the minimum value from a set of given numerical values. Let us group the relation student_details using the Group All operator, and store the result in the relation named student_group_all as shown below

```
grunt> student_group_all = Group student_details All;
```

It will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate the global minimum of GPA, i.e., minimum among the GPA values of all the students using the MIN() function as shown below.

```
grunt> student_gpa_min = foreach student_group_all  Generate
   (student_details.firstname, student_details.gpa), MIN(student_details.gpa);
```

Verification

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

```
grunt> Dump student_gpa_min;
```

Output

It will produce the following output, displaying the contents of the relation student_gpa_min.

```
(({(Bharathi),(Komal),(Archana),(Trupthi),(Preethi),(Rajesh),(siddarth),(Rajiv) } ,
   {    (72)    ,   (83)  ,     (87)   ,    (75)   ,   (93)   ,   (90)   ,     (78)   ,  (89)    }) ,72)
```

Apache Pig - Load & Store Functions:

The Load and Store functions in Apache Pig are used to determine how the data goes ad comes out of Pig. These functions are used with the load and store operators. Given below is the list of load and store functions available in Pig.

S.N.	Function & Description
1	PigStorage()To load and store structured files.
2	TextLoader()To load unstructured data into Pig.
3	BinStorage()To load and store data into Pig using machine readable format.

Apache Pig - PigStorage()

The PigStorage() function 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.

Syntax

```
grunt> PigStorage(field_delimiter)
```

Example

Let us suppose we have a file named student_data.txt in the HDFS directory named /data/ 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.
```

We can load the data using the PigStorage function as shown below.

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

In the above example, we have seen that we have used comma (‘,’) delimiter. Therefore, we have separated the values of a record using (,).

In the same way, we can use the PigStorage() function to store the data in to HDFS directory as shown below.

```
grunt> STORE student INTO ' /home/pig_Output/ ' USING PigStorage (',');
```

Apache Pig - TextLoader()

The Pig Latin function TextLoader() is a Load function which is used to load unstructured data in UTF-8 format.

Syntax

```
grunt> TextLoader()
```

Example
Let us assume there is a file with named stu_data.txt in the HDFS directory named /data/ as shown below.

```
001,Rajiv_Reddy,21,Hyderabad
002,siddarth_Battacharya,22,Kolkata
003,Rajesh_Khanna,22,Delhi
004,Preethi_Agarwal,21,Pune
005,Trupthi_Mohanthy,23,Bhuwaneshwar
006,Archana_Mishra,23,Chennai
007,Komal_Nayak,24,trivendram
008,Bharathi_Nambiayar,24,Chennai
```

Now let us load the above file using the TextLoader() function.

```
grunt> details = LOAD ‘/home/pig_data/stu_data.txt' USING TextLoader();
```

You can verify the loaded data using the Dump operator.

```
grunt> dump details;
```

```
(001,Rajiv_Reddy,21,Hyderabad)
(002,siddarth_Battacharya,22,Kolkata)
(003,Rajesh_Khanna,22,Delhi)
(004,Preethi_Agarwal,21,Pune)
(005,Trupthi_Mohanthy,23,Bhuwaneshwar)
(006,Archana_Mishra,23,Chennai)
(007,Komal_Nayak,24,trivendram)
(008,Bharathi_Nambiayar,24,Chennai)
```

Apache Pig - BinStorage()

The BinStorage() function is used to load and store the data into Pig using machine readable format. BinStorge() in Pig is generally used to store temporary data generated between the MapReduce jobs. It supports multiple locations as input.

Syntax

Given below is the syntax of the BinStorage() function.

```
grunt> BinStorage();
```

Example

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

Stu_data.txt

```
001,Rajiv_Reddy,21,Hyderabad
002,siddarth_Battacharya,22,Kolkata
003,Rajesh_Khanna,22,Delhi
004,Preethi_Agarwal,21,Pune
005,Trupthi_Mohanthy,23,Bhuwaneshwar
006,Archana_Mishra,23,Chennai
007,Komal_Nayak,24,trivendram
008,Bharathi_Nambiayar,24,Chennai
```

```
grunt> student_details = LOAD ‘/home/pig_data/stu_data.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, age:int, city:chararray);
```

Now, we can store this relation into the HDFS directory named /pig_data/ using the BinStorage() function.

```
grunt> STORE student_details INTO ‘/home/pig_Output/mydata' USING BinStorage();
```

Now, load the data from the file part-m-00000.

```
grunt> result = LOAD ‘/home/pig_Output/b/part-m-00000' USING BinStorage();
```

Verify the contents of the relation as shown below

```
grunt> Dump result;

(1,Rajiv_Reddy,21,Hyderabad)
(2,siddarth_Battacharya,22,Kolkata)
(3,Rajesh_Khanna,22,Delhi)
(4,Preethi_Agarwal,21,Pune)
(5,Trupthi_Mohanthy,23,Bhuwaneshwar)
(6,Archana_Mishra,23,Chennai)
(7,Komal_Nayak,24,trivendram)
(8,Bharathi_Nambiayar,24,Chennai)
```

Apache Pig - Bag & Tuple Functions

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

Apache Pig - TOBAG()

The TOBAG() function of Pig Latin converts one or more expressions to individual tuples. And these tuples are placed in a bag.

Syntax

```
TOBAG(expression [, expression ...])
```

Example

Assume we have a file named employee_details.txt in the HDFS directory /pig_data/, with the following content.

employee_details.txt

```
001,Robin,22,newyork
002,BOB,23,Kolkata
003,Maya,23,Tokyo
004,Sara,25,London
005,David,23,Bhuwaneshwar
006,Maggy,22,Chennai
```

We have loaded this file into Pig with the relation name emp_data as shown below.

```
grunt> emp_data = LOAD ‘/home/pig_data/employee_details.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, city:chararray);
```

Let us now convert the id, name, age and city, of each employee (record) into a tuple as shown below.

```
tobag = FOREACH emp_data GENERATE TOBAG (id,name,age,city);
```

You can verify the contents of the tobag relation using the Dump operator as shown below.

```
grunt> DUMP tobag;

({(1),(Robin),(22),(newyork)})
({(2),(BOB),(23),(Kolkata)})
({(3),(Maya),(23),(Tokyo)})
({(4),(Sara),(25),(London)})
({(5),(David),(23),(Bhuwaneshwar)})
({(6),(Maggy),(22),(Chennai)})
```

Apache Pig - TOP()

The TOP() function of Pig Latin is used to get the top N tuples of a bag. To this function, as inputs, we have to pass a relation, the number of tuples we want, and the column name whose values are being compared. This function will return a bag containing the required columns.

Syntax

```
grunt> TOP(topN,column,relation)
```

Example
Assume we have a file named employee_details.txt in the HDFS directory /pig_data/, with the following content.

employee_details.txt

```
001,Robin,22,newyork
002,BOB,23,Kolkata
003,Maya,23,Tokyo
004,Sara,25,London
005,David,23,Bhuwaneshwar
006,Maggy,22,Chennai
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

We have loaded this file into Pig with the relation name emp_data as shown below.

```
grunt> emp_data = LOAD ‘/home/pig_data/ employee_details.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, city:chararray);
```

Group the relation emp_data by age, and store it in the relation emp_group.

```
grunt> emp_group = Group emp_data BY age;
```

Verify the relation emp_group using the Dump operator as shown below.

```
grunt> Dump emp_group;

(22,{(12,Kelly,22,Chennai),(7,Robert,22,newyork),(6,Maggy,22,Chennai),(1,Robin, 22,newyork)})
(23,{(8,Syam,23,Kolkata),(5,David,23,Bhuwaneshwar),(3,Maya,23,Tokyo),(2,BOB,23, Kolkata)})
(25,{(11,Stacy,25,Bhuwaneshwar),(10,Saran,25,London),(9,Mary,25,Tokyo),(4,Sara, 25,London)})
```

Now, you can get the top two records of each group arranged in ascending order (based on id) as shown below.

```
grunt> data_top = FOREACH emp_group {
   top = TOP(2, 0, emp_data);
   GENERATE top;
}
```

In this example we are retriving the top 2 tuples of a group having greater id. Since we are retriving top 2 tuples basing on the id, we are passing the index of the column name id as second parameter of TOP() function.

You can verify the contents of the data_top relation using the Dump operator as shown below.

```
grunt> Dump data_top;

({(7,Robert,22,newyork),(12,Kelly,22,Chennai)})
({(5,David,23,Bhuwaneshwar),(8,Syam,23,Kolkata)})
({(10,Saran,25,London),(11,Stacy,25,Bhuwaneshwar)})
```