UNIT – 2

Pig

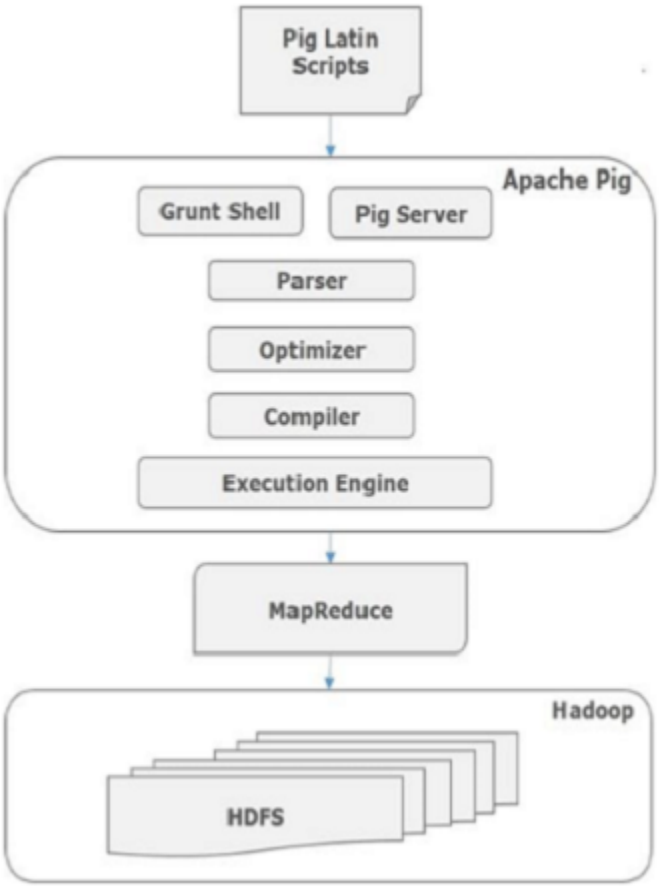
What is Apache Pig?

- The Apache Pig is an abstraction over the MapReduce. Using this tool we can analyze a large set of data as the data flow.
- Pig provides a high-level programming language called **Pig Latin**, which is used to write code to read, write, process data.
- All scripts in Pig Latin are internally converted to Map and Reduce tasks. Pig has another component, known as **Pig Engine.** It takes the Pig Latin code as input and converts those script to MapReduce task.

What are the Features of Apache Pig?

- Apache Pig comes with the following features –
  - ✓ It provides many operators to perform operations like join, sort, filer, etc.
  - ✓ Pig Latin is similar to SQL and it is easy to write a Pig script using the SQL concept.
  - ✓ The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
  - ✓ Using the existing operators, users can develop their own functions to read, process, and write data.
  - ✓ Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Pig Architecture



- **The Apache Pig Architecture:**
  - 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 has shown above.

➢ **Components of Pig:**

- In the figure we have seen there are different components in the Pig.
- **Parser**: The Pig scripts are initially going to the Parser. It performs different types of checking process like syntax check etc. After that, it creates a Directed Acyclic Graph (DAG), which represents the Pig Latin statements and logical operators. In DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.
- **Optimizer**: The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushes down.

➢ **Components of Pig:**

- **Compiler**: The compiler compiles the optimized logical plan into a series of MapReduce jobs.
- **Execution Engine**: Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

Pig Latin Data Model

Bag

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by '{}'. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contains the same number of fields or that the fields in the same position (column) have the same type.

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Field

A piece of data or a simple atomic value is known as a field.

Data type	Discription	Example
int	Represents a signed 32-bit integer.	8
long	Represents a signed 64-bit integer.	5L
float	Represents a signed 32-bit integer.	5.5L
double	Represents a signed 64-bit integer.	10.5L

Data type	Description	Example
Chararray	Represents a character array (string) in Unicode UTF-8 format.	'Asterix Solution'
Bytearray	Represents a Byte array (blob).	
Boolean	Represents a Boolean values.	ture/false

When you do not assign any data type while making your application it will be by default blob file or Bytearray

Data type	Description	Example
Datetime	Represents a Date-time.	1970-01-01 T00:00:00.000 +00:00
Biginteger	Represents a java Biginteger.	
Bigdecimal	Represents a Bigdecimal.	185.9837 62527286 93883

Data type	Description	Example
Tuple	A tuple is an ordered set of fields.	(raja,30)
Bag	A bag is a collection of tuples.	{(raju,30), (Mohhammad,45)}
Map	A Map is a set of key-value pairs.	['name'#'Raju', 'age'#30]

➢ **Relational Operations in Pig Latin:**

- The following table describes the relational operators of Pig Latin.

Operator	Description
LOAD	Load data from HDFS or local disk
STORE	Save data to HDFS or local disk
FILTER	Remove unwanted rows
DISTINCT	Remove duplicate rows
FOREACH, GENERATE	To generate data transformations based on columns of data.
STREAM	Transform relation with external program
DUMP	Print data of a relation

Relational Operations in Pig Latin	
Operator	**Description**
JOIN	To join two or more relations.
COGROUP	Group the data in two or more relation
GROUP	To group the data in a single relation.
CROSS	Cross product of tables
ORDER	Arrange relation in sorted order.
LIMIT	To get limited number of tuples.
UNION	Combine multiple relations into one
SPLIT	Split one relation to multiple relations
DESCRIBE	Describe the schema of a relation.

And other Operators are **Comparison Operators** One important from them

2. Pattern Matching Operators (Regular Expressions)

Operator	Description	Example
`matches`	Checks if a string matches a regex pattern	`name matches 'A.*'`

- **Example Usage** (Filtering names starting with "A")

pig Copy Edit

```
students = LOAD 'students.txt' AS (id: INT, name: CHARARRAY);
a_students = FILTER students BY name matches 'A.*';
DUMP a_students;
```

Arithmetic Operators

+,-,*,/,%,

And

? :	**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 Built-in Functions

Apache Pig provides various built-in functions namely **eval, load, store, math, string, bag** and **tuple** functions.

Eval Functions

Given below is the list of **eval** functions provided by Apache Pig

S.N.	Function & Description
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	**00000TOKENIZE()** 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 - User Defined Functions

In addition to the built-in functions, Apache Pig provides extensive support for **U**ser **D**efined **F**unctions (UDFs). Using these UDFs, we can define our own functions and use them. The UDF support is provided in six programming languages, namely, Java, Jython, Python, JavaScript, Ruby and Groovy.

For writing UDFs, complete support is provided in Java and limited support is provided in all the remaining languages. Using Java, you can write UDFs involving all parts of the processing like data load/store, column transformation, and aggregation. Since Apache Pig has been written in Java, the UDFs written using Java language work efficiently compared to other languages.

In Apache Pig, we also have a Java repository for UDFs named **Piggybank**. Using Piggybank, we can access Java UDFs written by other users, and contribute our own UDFs.

Types of UDFs in Java

While writing UDFs using Java, we can create and use the following three types of functions −

- **Filter Functions** − The filter functions are used as conditions in filter statements. These functions accept a Pig value as input and return a Boolean value.
- **Eval Functions** − The Eval functions are used in FOREACH-GENERATE statements. These functions accept a Pig value as input and return a Pig result.
- **Algebraic Functions** − The Algebraic functions act on inner bags in a FOREACHGENERATE statement. These functions are used to perform full MapReduce operations on an inner bag.

Writing UDFs using Java

To write a UDF using Java, we have to integrate the jar file **Pig-0.15.0.jar**. In this section, we discuss how to write a sample UDF using Eclipse. Before proceeding further, make sure you have installed Eclipse and Maven in your system.

Follow the steps given below to write a UDF function −

- Open Eclipse and create a new project (say **myproject**).
- Convert the newly created project into a Maven project.
- Copy the following content in the pom.xml. This file contains the Maven dependencies for Apache Pig and Hadoop-core jar files.

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0http://maven.apache
.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>Pig_Udf</groupId>
  <artifactId>Pig_Udf</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.3</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
```

```
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pig</artifactId>
      <version>0.15.0</version>
    </dependency>

    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-core</artifactId>
      <version>0.20.2</version>
    </dependency>

  </dependencies>

</project>
```

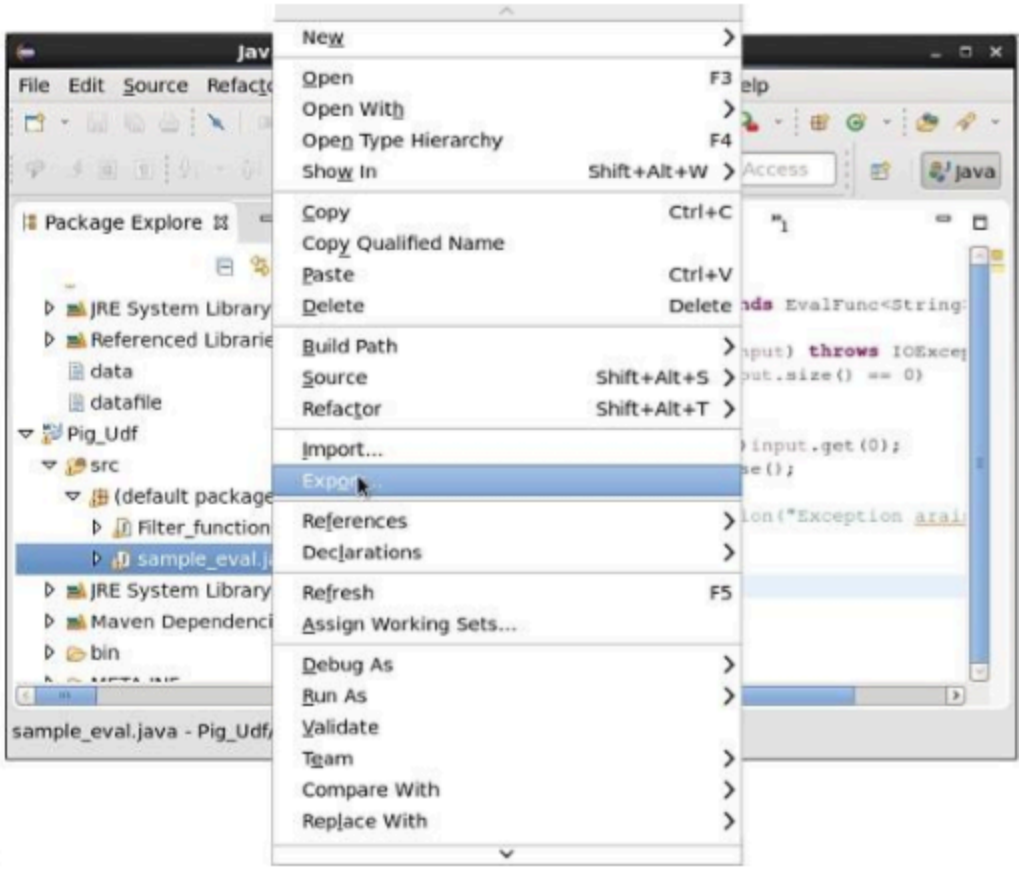
- Save the file and refresh it. In the **Maven Dependencies** section, you can find the downloaded jar files.
- Create a new class file with name **Sample_Eval** and copy the following content in it.

```
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;

import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
```

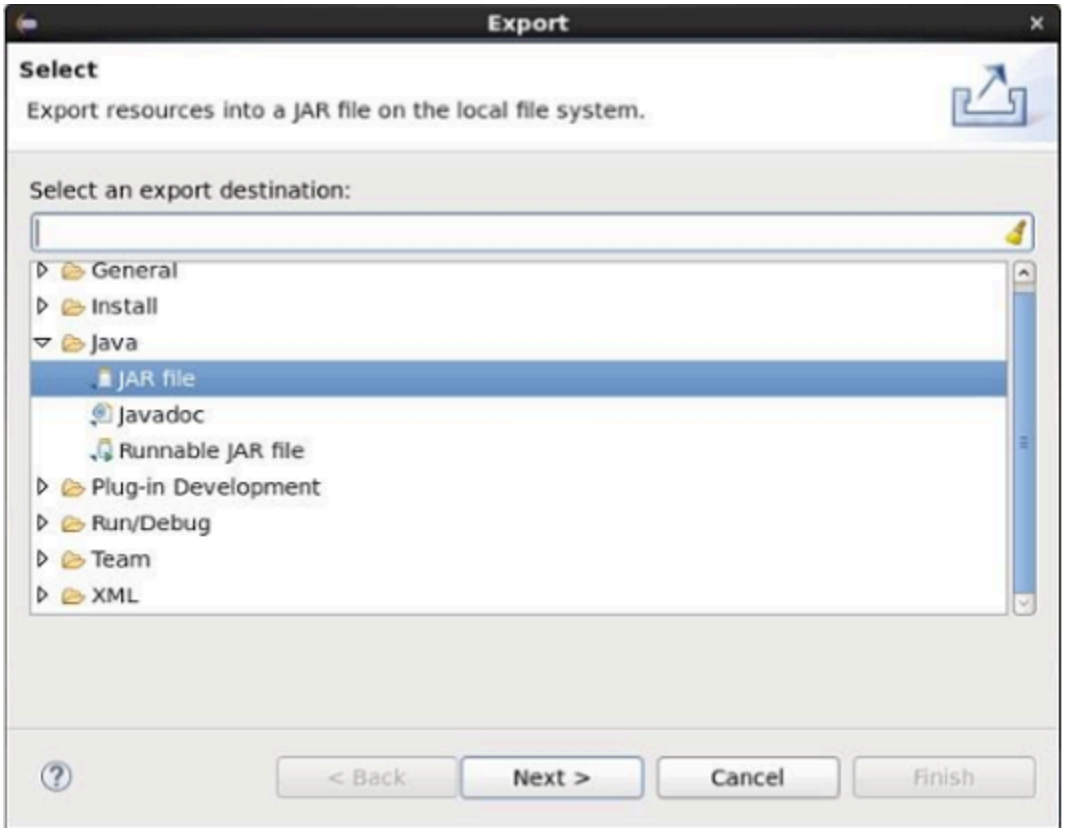
```
public class Sample_Eval extends EvalFunc<String>{


  public String exec(Tuple input) throws IOException {
    if (input == null || input.size() == 0)
    return null;
    String str = (String)input.get(0);
    return str.toUpperCase();
  }
}
```

While writing UDFs, it is mandatory to inherit the EvalFunc class and provide implementation to **exec()** function. Within this function, the code required for the UDF is written. In the above example, we have return the code to convert the contents of the given column to uppercase.

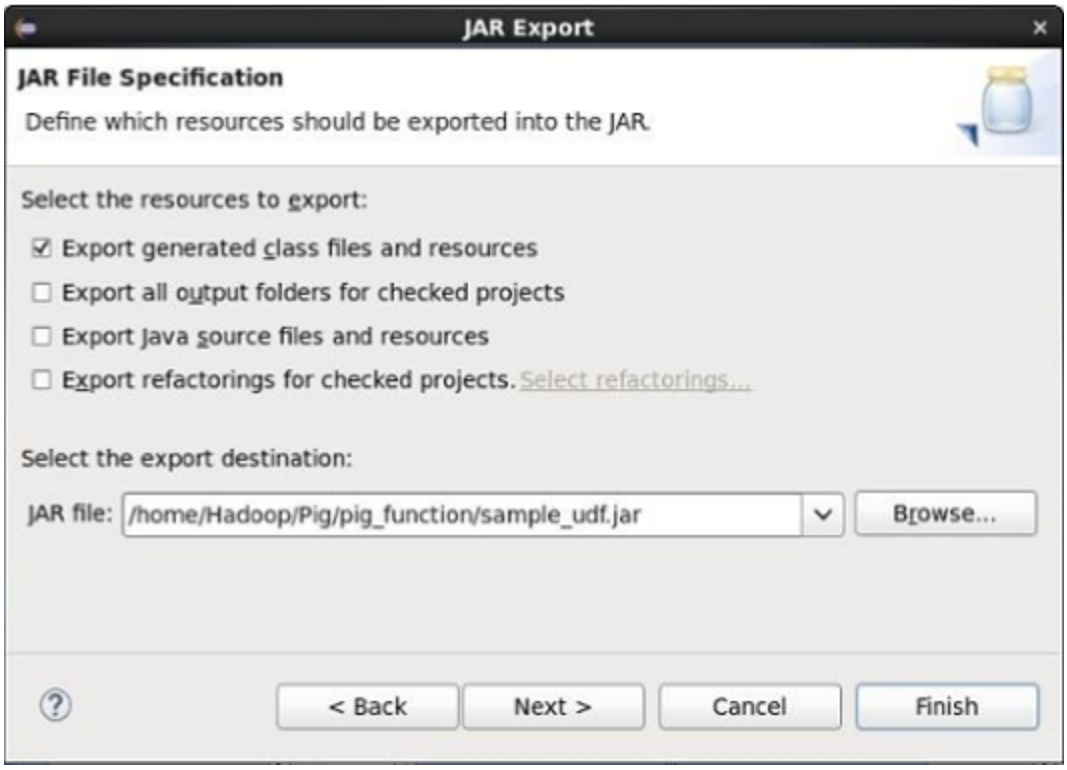
- After compiling the class without errors, right-click on the Sample_Eval.java file. It gives you a menu. Select **export** as shown in the following screenshot.



- On clicking **export**, you will get the following window. Click on **JAR file**.



- Proceed further by clicking **Next>** button. You will get another window where you need to enter the path in the local file system, where you need to store the jar file.



- Finally click the **Finish** button. In the specified folder, a Jar file **sample_udf.jar** is created. This jar file contains the UDF written in Java.

Using the UDF

After writing the UDF and generating the Jar file, follow the steps given below –

Step 1: Registering the Jar file

After writing UDF (in Java) we have to register the Jar file that contain the UDF using the Register operator. By registering the Jar file, users can intimate the location of the UDF to Apache Pig.

Syntax

Given below is the syntax of the Register operator.

REGISTER path;

Example

As an example let us register the sample_udf.jar created earlier in this chapter.

Start Apache Pig in local mode and register the jar file sample_udf.jar as shown below.

$cd PIG_HOME/bin

$./pig x local

REGISTER '/$PIG_HOME/sample_udf.jar'

Note − assume the Jar file in the path − /$PIG_HOME/sample_udf.jar

Step 2: Defining Alias

After registering the UDF we can define an alias to it using the **Define** operator.

Syntax

Given below is the syntax of the Define operator.

DEFINE alias {function | [`command` [input] [output] [ship] [cache] [stderr]] };

Example

Define the alias for sample_eval as shown below.

DEFINE sample_eval sample_eval();

Step 3: Using the UDF

After defining the alias you can use the UDF same as the built-in functions. Suppose there is a file named emp_data in the HDFS **/Pig_Data/** directory with the following content.

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

And assume we have loaded this file into Pig as shown below.

```
grunt> emp_data = LOAD 'hdfs://localhost:9000/pig_data/emp1.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, city:chararray);
```

Let us now convert the names of the employees in to upper case using the UDF **sample_eval**.

```
grunt> Upper_case = FOREACH emp_data GENERATE sample_eval(name);
```

Verify the contents of the relation **Upper_case** as shown below.

```
grunt> Dump Upper_case;
```

OUTPUT:

```
(ROBIN)
(BOB)
(MAYA)
(SARA)
(DAVID)
(MAGGY)
(ROBERT)
(SYAM)
(MARY)
(SARAN)
(STACY)
(KELLY)
```

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.

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

Executing a Pig Script from HDFS

We can also 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 **/pig_data/**. We can execute it as shown below.

$ pig -x mapreduce hdfs://localhost:9000/pig_data/Sample_script.pig

Example

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

We also have a sample script with the name **sample_script.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;
```

- The first statement of the script will load the data in the file named **student_details.txt** as a relation named **student**.
- The second statement of the script will arrange the tuples of the relation in descending order, based on age, and store it as **student_order**.
- The third statement of the script will store the first 4 tuples of **student_order** as **student_limit**.

- Finally the fourth statement will dump the content of the relation **student_limit**.

Let us now execute the **sample_script.pig** as shown below.

$./pig -x mapreduce hdfs://localhost:9000/pig_data/sample_script.pig

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)