Birla Vishvakarma Mahavidyalaya
Engineering College, Vallabh Vidyanagar

4CP02: DATA ANALYTICS AND VISUALIZATION

Unit 3

Map-Reduce Programming with Apache Hadoop

Apache Pig

- Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.
- It raises the level of abstraction for processing large datasets.
- MapReduce allows programmers to specify a map function followed by a reduce function, but working out how to fit your data processing into this pattern can be a challenge, when it requires multiple MapReduce stages.

Apache Pig

- With Pig, the data structures are much richer, typically being multivalued and nested, and the transformations you can apply to the data are much more powerful.
- Pig is made up of two pieces:
  - The language used to express data flows, called Pig Latin.
  - The execution environment to run Pig Latin programs.
- There are currently two environments: local execution in a single JVM and distributed execution on a Hadoop cluster.

Apache Pig

- A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input data to produce output.
- Taken as a whole, the operations describe a data flow, which the Pig execution environment translates into an executable representation and then runs.
- Under the covers, Pig turns the transformations into a series of MapReduce jobs.
- Pig is a scripting language for exploring large datasets.

Apache Pig

- One criticism of MapReduce is that the development cycle is very long.
- Writing the mappers and reducers, compiling and packaging the code, submitting the job(s), and retrieving the results is a time consuming business.
- Pig's sweet spot is its ability to process terabytes of data in response to a half-dozen lines of Pig Latin issued from the console.
- Indeed, it was created at Yahoo! to make it easier for researchers and engineers to mine the huge datasets.

Apache Pig

- Pig was designed to be extensible. Virtually all parts of the processing path are customizable: loading, storing, filtering, grouping, and joining can all be altered by user-defined functions (UDFs).
- As another benefit, UDFs tend to be more reusable than the libraries developed for writing MapReduce programs.

Installing and Running Apache Pig

- Download a stable release from http://pig.apache.org/releases.html, and unpack the tarball in a suitable place on your system.
- Set environment variables path.
- In local mode, Pig runs in a single JVM and accesses the local filesystem.
- This mode is suitable only for small datasets and when trying out Pig.

```
% pig -x local
```

- This starts Grunt, the Pig interactive shell.

Installing and Running Apache Pig

- In MapReduce mode, Pig translates queries into MapReduce jobs and runs them on a Hadoop cluster.
- The cluster may be a pseudo- or fully distributed cluster.
- Once you have configured Pig to connect to a Hadoop cluster, you can launch Pig.
- Once you have configured Pig to connect to a Hadoop cluster, you can launch Pig: Script, Grunt, Embedded

Installing and Running Apache Pig

- Script: Pig can run a script file that contains Pig commands. For example, `pig script.pig` runs the commands in the local file script.pig
- Grunt: Grunt is an interactive shell for running Pig commands. Grunt is started when no file is specified for Pig to run.
- Embedded: You can run Pig programs from Java using the PigServer class, much like you can use JDBC to run SQL programs from Java.
- For programmatic access to Grunt, use PigRunner.

Pig Latin

- Pig Latin Statements are generally ordered as follows:
- 1. LOAD statement that reads data from the file system.
- 2. Series of statements to perform transformations.
- 3. DUMP or STORE to display/store result.
- Single line Comments: --
- Multiline Comments: /* */

Pig Data Types – Primitive and Complex

Primitive Data Types: also called as simple datatypes.

- **int** : signed 32 bit integer
- **long** : 64 bit signed integer
- **float** : 32 bit floating point
- **double** : 64 bit floating point
- **chararray** : It is character array in unicode UTF-8 format. This corresponds to java's String object.
- **bytearray** : Used to represent bytes. It is the default data type. If you don't specify a data type for a filed, then bytearray datatype is assigned for the field.
- **boolean** : to represent true/false values.

Complex Types:

Pig supports three complex data types. They are listed below:

- **Tuple** : An ordered set of fields. Tuple is represented by braces. Example: (1,2)
- **Bag** : A set of tuples is called a bag. Bag is represented by flower or curly braces. Example: {(1,2),(3,4)}
- **Map** : A set of key value pairs. Map is represented in a square brackets. Example: [key#value] . The # is used to separate key and value.

Field: A field is a piece of data. In the above data set product_name is a field.

Relation: Relation represents the complete database. A relation is a bag. To be precise relation is an outer bag. We can call a relation as a bag of tuples.

Creating Schema, Reading and Writing Data

Consider the following products data set in Hadoop as an example:

10, iphone, 1000

20, samsung, 2000

30, nokia, 3000

Here first field is the product id, second field is the product name and third field is the product price.

Defining Schema:

- The LOAD operator is used to define a schema for a data set.

1. Creating Schema without specifying any fields.

```
grunt> A = LOAD '/user/hadoop/products';
```

Pig is a data flow language.

Each operational statement in pig consists of a relation and an operation.

The left side of the statement is called relation and the right side is called the operation.

Pig statements must terminated with a semicolon.

Here A is a relation. /user/hadoop/products is the file in the hadoop.

To view the schema of a relation,

```
grunt> describe A;
Schema for A unknown.
```

As there are no fields are defined, the above describe statement on A shows that “Schema for A unkown”. To display the contents on the console use the DUMP operator.

```
grunt> DUMP A;
(10,iphone,1000)
(20,samsung,2000)
(30,nokia,3000)
```

To write the data set into HDFS, use the STORE operator as shown below

```
grunt> STORE A INTO 'hadoop directory name'
```

2. . Defining schema without specifying any data types.

We can create a schema just by specifying the field names without any data types. An example is shown below:

grunt> A = LOAD '/user/hadoop/products' USING PigStorage(',') AS (id, product_name, price);

grunt> describe A;

A: {id: bytearray,product_name: bytearray,price: bytearray}

grunt> STORE A into '/user/hadoop/products' USING PigStorage('|'); --Writes data with pipe as delimiter into hdfs product directory.

3. **Defining schema with field names and data types.**

To specify the data type use the colon. Take a look at the below example:

```
grunt> A = LOAD '/user/hadoop/products' USING PigStorage(',') AS (id:int,
product_name:chararray, price:int);

grunt> describe A;
A: {id: int,product_name: chararray,price: int}
```

4. **Accessing the Fields**:

The fields can be accessed in two ways:

- **Field Names**: We can specify the field name to access the values from that particular value.
- **Positional Parameters**: The field positions start from 0 to n. $0 indicates first field, $1 indicates second field.

Example:

```
grunt> A = LOAD '/user/products/products' USING PigStorage(',') AS (id:int,
product_name:chararray, price:int);
grunt> B = FOREACH A GENERATE id;
grunt> C = FOREACH A GENERATE $1,$2;
grunt> DUMP B;
(10)
(20)
(30)
grunt> DUMP C;
(iphone,1000)
(samsung,2000)
(nokia,3000)
```

How to Filter Records

- The Filter functionality is similar to the WHERE clause in SQL.
- The FILTER operator in pig is used to remove unwanted records from the data file.
- The syntax of FILTER operator is shown below:

<new relation> = FILTER <relation> BY <condition>

- Here relation is the data set on which the filter is applied, condition is the filter condition and new relation is the relation created after filtering the rows.

Pig Filter Examples:

- Lets consider the below sales data set as an example

year,product,quantity

2000, iphone, 1000

2001, iphone, 1500

2002, iphone, 2000

2000, nokia, 1200

2001, nokia, 1500

2002, nokia, 900

1. select products whose quantity is greater than or equal to 1000.

```
grunt> A = LOAD '/user/hadoop/sales' USING PigStorage(',') AS
(year:int,product:chararray,quantity:int);

grunt> B = FILTER A BY quantity >= 1000;

grunt> DUMP B;

(2000,iphone,1000)

(2001,iphone,1500)

(2002,iphone,2000)

(2000,nokia,1200)

(2001,nokia,1500)
```

2. select products whose quantity is greater than 1000 and year is 2001

grunt> C = FILTER A BY quantity > 1000 AND year == 2001;
(2001,iphone,1500)
(2001,nokia,1500)

3. select products with year not in 2000

grunt> D = FILTER A BY year != 2000;
grunt> DUMP D;
(2001,iphone,1500)
(2002,iphone,2000)
(2001,nokia,1500)
(2002,nokia,900)

You can use all the logical operators (NOT, AND, OR) and relational operators (< , >, ==, !=, >=, <=) in the filter conditions.

Word Count in Pig Latin

Assume we have data in the file like below.

This is a hadoop post
hadoop is a bigdata technology

and we want to generate output for count of each word like below

(a,2)
(is,2)
(This,1)
(class,1)
(hadoop,2)
(bigdata,1)
(technology,1)

1.Load the data from HDFS

Use Load statement to load the data into a relation .
As keyword used to declare column names, as we dont have any columns, we declared only one column named line.

input = LOAD '/path/to/file/' AS(line:Chararray);

2. Convert the Sentence into words.

The data we have is in sentences. So we have to convert that data into words using TOKENIZE Function.

(TOKENIZE(line));

(or)

If we have any delimeter like space we can specify as

(TOKENIZE(line,' '));

Output will be like this:

({(This),(is),(a),(hadoop),(class)})
({(hadoop),(is),(a),(bigdata),(technology)})

but we have to convert it into multiple rows like below

(This)
(is)
(a)
(hadoop)
(class)
(hadoop)
(is)
(a)
(bigdata)
(technology)

3.Convert Column into Rows

we have to convert every line of data into multiple rows ,for this we have function called FLATTEN in pig.

Using FLATTEN function the bag is converted into tuple, means the array of strings converted into multiple rows.

Words = FOREACH input GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;

Then the ouput is like below

(This)
(is)
(a)
(hadoop)
(class)
(hadoop)
(is)
(a)
(bigdata)
(technology)

3. Apply GROUP BY

We have to count each word occurance, for that we have to group all the words.

Grouped = GROUP words BY word;

4. Generate word count

wordcount = FOREACH Grouped GENERATE group, COUNT(words);

We can print the word count on console using Dump.

DUMP wordcount;

Output will be like below.

(a,2)
(is,2)
(This,1)
(class,1)
(hadoop,2)
(bigdata,1)
(technology,1

Writing Macro in Pig Latin

We will take sample emp data like below.

eno,ename,sal,dno

10,Balu,10000,15

15,Bala,20000,25

30,Sai,30000,15

40,Nirupam,40000,35

using above data I would like to have employee data who belong to department number 15.

1. Example without Macro

Write below code in a file called filterwitoutmacro.pig

```
emp = load '/data/employee'using PigStorage(',') as (eno,ename,sal,dno);
empdno15 =filter emp by $3==15;
dump empdno15;
```

run pig latin code from file.

pig -f /path/to/filterwitoutmacro.pig

2. Same example with macro

2.1 Create a macro

```
DEFINE myfilter(relvar,colvar) returns x{
$x = filter $relvar by $colvar==15;
};
```

Above macro takes two values as input,one is relation variable (relvar) and second is column variable (colvar)

2.2 Usage of macro

we can use myfilter macro like below.

```
emp = load '/data/employee'using PigStorage(',') as (eno,ename,sal,dno);
empdno15 =myfilter( emp,dno);
dump empdno15;
```

we can write macro creation code and macro usage code in same file ,can run file with -f option.

```
pig -f /path/to/myfilterwithembeddedmacro.pig
```

3. same example with external macro.

3.1 write above macro in separate file called myfilter.macro

```
--myfilter.macro
DEFINE myfilter(relvar,colvar) returns x{
$x = filter $relvar by $colvar==15;
}
```

3.2 Import macro file in another pig latin script file.

```
IMPORT '/path/to/myfilter.macro'
emp = load '/data/employee'using PigStorage(',') as (eno,ename,sal,dno);
empdno15 =myfilter( emp,dno);
dump empdno15;
```

and we run pig latin script file using -f option.

```
pig -f /path/to/myfilterwithexternalmacro.pig
```

Apache Pig - User Defined Functions (UDF)

- The UDF support is provided in six programming languages, namely, Java, Jython, Python, JavaScript, Ruby and Groovy.
- For writing UDF’s, complete support is provided in Java and limited support is provided in all the remaining languages.
- Using Java, you can write UDF’s involving all parts of the processing like data load/store, column transformation, and aggregation.
- Since Apache Pig has been written in Java, the UDF’s written using Java language work efficiently compared to other languages.

Types of UDF’s in Java

- While writing UDF’s 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.

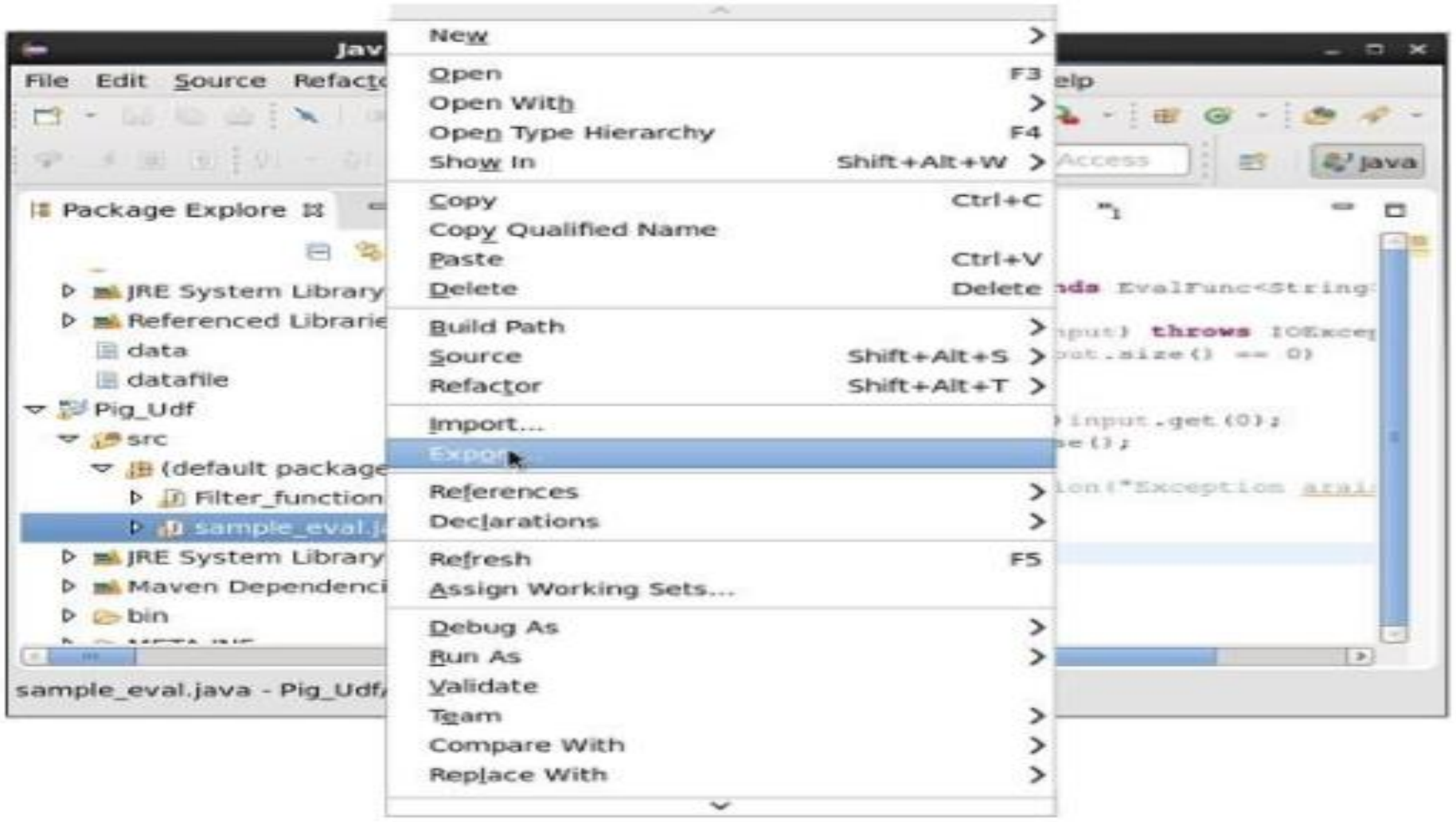
Writing UDF’s using Java

Follow the steps given below to write a UDF function −

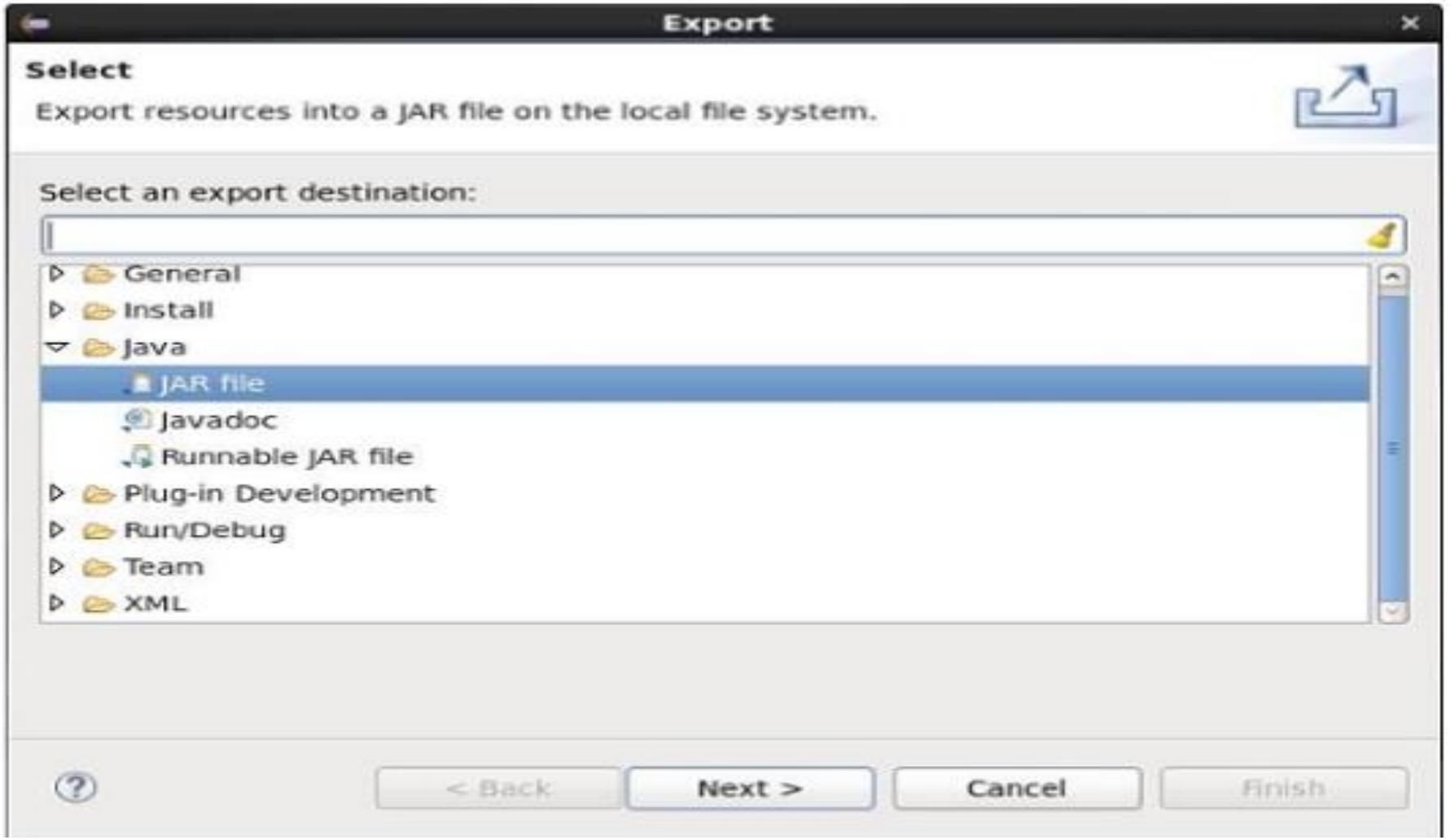
- Open Eclipse and create a new project (say **myproject**).
- Convert the newly created project into a Maven project.
- Copy the following content in the pom.xml. This file contains the Maven dependencies for Apache Pig and Hadoop-core jar files.
- Save the file and refresh it. In the **Maven Dependencies**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

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();
  }
}
```

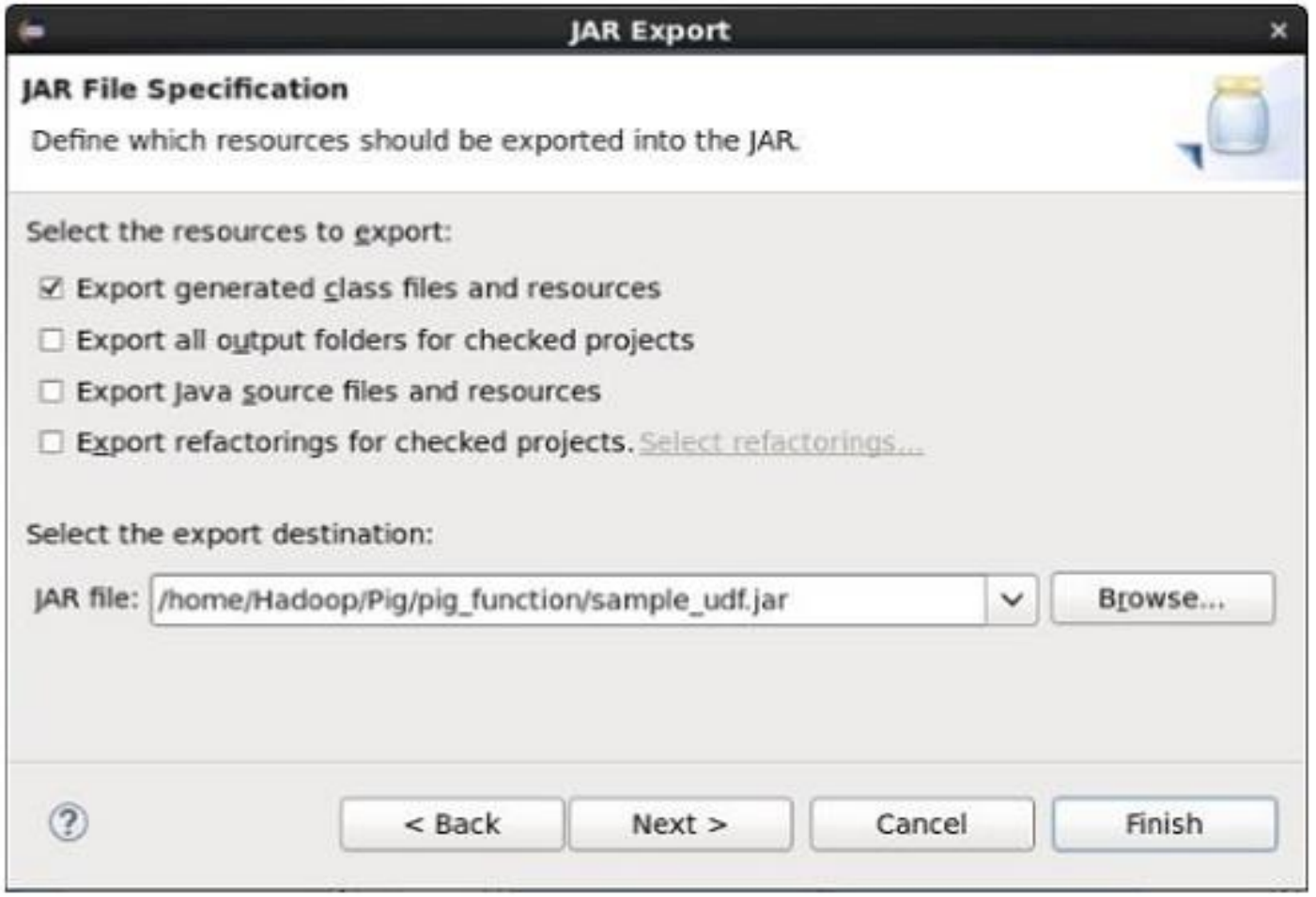
After compiling the class without errors, right-click on the Sample_Eval.java file. It gives you a menu. Select export as shown in the following screenshot.



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.
- 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.

- 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.

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);
```

```
grunt> Dump Upper_case;

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

Features of Pig

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

Apache Pig VS MapReduce

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

Pig VS SQL

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

Pig Latin: Exercise

- Word Count Job
- Writing Macro
- Writing UDF