Class 06 Advanced Pig

TOPICS COVERED

❍ **Relational Operators in Pig**

CLASS OBJECTIVES

At the end of this class you will be able to:

⏭ Use relational operators in Pig to transform data

TOPIC

1 Relational Operators in Pig

TOPIC OBJECTIVES

At the end of this topic you will be able to:

⏭ Use relational operators in Pig

Introduction

You have now been made familiar with scripting in Pig Latin. Class 6 is intended to be more of a hands-on class to demonstrate you implementation of some of the commonly used relational operators. These also include some more operators such as COGROUP, UNION and RANK.

You can use **Relational operators** in Pig Latin to transform data by filtering, grouping, sorting, and joining. The basic relational operators are as follows:

After this statement add the following in a bulleted list in the next line:

- FOREACH
- FILTER
- GROUP
- ORDER BY
- DISTINCT
- JOIN
- LIMIT
- SAMPLE

In this class you will see implementations of these basic relational operators.

FOREACH

On the basis of a predefined expression, the FOREACH operator iterates every record to perform a transformation. After each application of the supplied expression, the FOREACH operator produces a new set of records. The syntax of the FOREACH operator is:

```
alias = FOREACH {block | nested_block};
```

Table 1 presents some common terms of the FOREACH operator:

TABLE 1
FOREACH Operator's Common Terms

Terms	Description
alias	Describes the name of the relation
expression	Describes an expression in the FOREACH operator
nested_alias	Describes the name of the inner bag
nested_exp	Describes some arbitrary, supported expression in the FOREACH operator

The following script illustrates an example of the FOREACH operator:

```
employee = load '/home/titan/hadoop/pig/pig-0.12.0/data' using
PigStorage('');
result = foreach employee generate *;
dump result;
(raj,2,1.1)
(manam,3,4.3)
(bindhu,3,1.1F)
(pavan,6,3.1)
```

The example uses the relations **employee** and **result**. Both the relations are identical. The example uses the special symbol asterisk (*) to project all fields that relate the relation employee to relation result.

In this example, the FOREACH operator projects two fields from the relation employee to relation result as follows:

```
employee = load '/home/titan/hadoop/pig/pig-0.12.0/data' using
PigStorage('');
result = foreach employee generate name,id;
dump result;
(raj,2)
(vina,6)
(raju,7)
(jay,9)
```

FILTER

The FILTER statement contains a predicate that helps you in choosing the records that need to be retained in the pipeline. The predicate defined in the FILTER statement should be true for a record for it to pass down the pipeline successfully. The syntax of the FILTER statement is:

```
alias = FILTER alias BY expression;
```

Table 2 provides some common terms of the FILTER operator:

TABLE 2
FILTER Terms

Terms	Description
Alias	Describes the name of the relation
BY	Describes a required key word
Expression	Describes a Boolean expression

The following script illustrates an example of the FILTER statement:

```
a = load '/home/titan/hadoop/pig/pig-0.12.0/students' AS
(a1:int,a2:int,a3:int);
dump a;
(1,2,2)
(4,2,1)
(6,1,3)
(6,3,2)
Z = filter a by a3 == 2;
dump Z;
(1,2,2)
(6,3,2)
B = FILTER a BY (a1 == 6) OR (NOT (a2+a3 > a1));
DUMP X;
(4,2,1)
(6,1,3)
(6,3,2)
```

Knowledge Check—1

1. Using the Pig script, write the code to load one text file into a variable and load another text file into another variable. Then, 'JOIN' these two variables by their first fields and display the result.

GROUP

Pig Latin provides different operators for group and aggregate functions. The GROUP statement in Pig Latin has a SQL-like syntax, but differs in functionality compared to the SQL GROUP BY clause. In Pig Latin, the GROUP statement is used to group the data in one or more relations. The GROUP BY clause in SQL creates a group that must feed directly into one or more aggregate functions. A star expression cannot be included in a GROUP BY column in Pig Latin. The syntax of the GROUP statement is:

```
alias = GROUP alias { ALL | BY expression} [, alias ALL | BY
expression…] [USING 'collected' | 'merge'] [PARTITION BY partitioner]
[PARALLEL n];
```

ADDITIONAL KNOW–HOW

The GROUP and COGROUP operators are identical. Both operators work with one or more relations. For readability, the GROUP operator is used in statements involving one relation, and the COGROUP operator is used in statements involving two or more relations. You can COGROUP up to, but no more than, 127 relations at a time.

Table 3 describes some common terms of the GROUP operator:

TABLE 3
The GROUP Operator's Common Terms

Terms	Description
alias	Describes the name of a relation
ALL	Signifies a key word that is used to input all tuples to a single group while performing aggregates across entire relations Syntax: B = GROUP A ALL;
BY	Signifies a key word that is used to group the relation by field, tuple, or expression Syntax: B = GROUP A BY f1;
PARTITION BY partitioner	Describes the Hadoop Partitioner to control the partitioning of the keys of the intermediate map-outputs

EXAMPLE

You group relation A on the fields 'name,' 'age,' and 'gpa' to form relation B. You can use the DESCRIBE and ILLUSTRATE operators to examine the structure of the relation.

```
A = load '/user/pig/tests/data/singlefile/studenttab10k' using
PigStorage('\t') as (name, age, gpa);
describe A;
A: {name: chararray, age: int, gpa: float}
dump A;
(Joy,21,3.09)
(Mayuri,23,9.8F)
(raj,20,7.90)
grunt> A = load '/user/pig/tests/data/singlefile/studenttab10k' using
PigStorage('\t') as (name, age, gpa);
grunt> B = group A by name;
grunt> C = foreach B generate group, COUNT(A.$1);
grunt> illustrate C;
----------------------------------------
| A | name | age | gpa |
----------------------------------------
| | xavi ... bec | 58 | 2.99 |
| | xavi ... bec | 23 | 0.59 |
----------------------------------------
----------------------------------------------------------------------
----------
| B | group | A: (name, age, gpa ) |
----------------------------------------------------------------------
----------
| | xavi ... bec | {(xavi ... bec, 58, 2.99), (xavi ... bec, 23,
0.59)} |
----------------------------------------------------------------------
----------
---------------------------------
| C | group | count1 |
---------------------------------
| | xavi ... bec | 2 |
---------------------------------
```

It can be seen in these FOREACH statements that you can refer to the fields in relation B by the names "group" and "A" or by a positional notation.

```
grunt>A = load 'mydata' using PigStorage()
as (a, b, c);
grunt>B = group A by a;
grunt>C = foreach B {
D = distinct A,b;
generate flatten(group), COUNT(D);
}
X = foreach B generate $0, $1.name;

dump X;
(21,{(raj),(Joy)})
(23,{(Mayuri)})
```

Order By

In Pig Latin, the ORDER statement is used to sort relation based on one or more fields. A state of **Total Order** is achieved when the data in all the partitions is stored. Total Order indicates that for all n, there are n-l records present in all partitions of data. The syntax of the ORDER statement is:

```
alias = ORDER alias BY { * [ASC|DESC] | field_alias [ASC | DESC] [,
field_alias [ASC|DESC]…] } [PARALLEL n];
```

Table 4 describes some common terms of the ORDER operator:

TABLE 4 Common Terms of the ORDER Operator

Term	Description
alias	Describes the name of a relation
*	Describes a tuple designator
field_alias	Describes a field in the relation
ASC	Sorts that data in ascending order.
DESC	Sorts the data in descending order.

ORDER BY is NOT stable. If multiple records have the same ORDER BY key, the order in which these records are returned is not defined and is not guaranteed to be the same from one run to the next.

TECHNICAL STUFF

Pig supports ordering on fields with simple types or by tuple designator (*). You cannot order on fields with complex types or by expressions. The following commands shows the syntax followed while ordering fields in Pig:

```
loading = LOAD 'mydata' AS (x: int, y: map[]);
ord = ORDER loading BY x; -- this is allowed because x is a simple type
ord = ORDER loading BY y; -- this is not allowed because y is a complex
type
ord = ORDER loading BY y#'id'; -- this is not allowed because y#'id' is
an expression
```

Following scripts depict some examples of order by statement in Pig:

```
data = load '/home/titan/hadoop/pig/pig-0.12.0/students' AS
(a1:int,a2:int,a3:int);
dump data;
(1,2,4)
(2,4,1)
(4,2,5)
(3,5,7)
cal = ORDER data BY $0;
C = LIMIT cal 3;
DUMP C;
(3,5,7)
(4,2,5)
(1,2,4)
(2,4,1)
```

DISTINCT

In Pig Latin, the DISTINCT statement operates on the entire records and not on individual fields. The DISTINCT statement is used to remove duplicate fields from the records. The syntax of the DISTINCT statement is as follows:

```
alias = DISTINCT alias [PARTITION BY partitioner] [PARALLEL n];
```

Table 5 presents some common terms of the DISTINCT operator:

TABLE 5 Common Terms of the DISTINCT Operator

Term	Description
alias	Describes the name of the relation
PARTITION BY partitioner	Describes the Hadoop Partitioner to control the partitioning of the keys of the intermediate map outputs.

Following scripts depict some examples of distinct statement in Pig

```
stud = load '/home/titan/hadoop/pig/pig-0.12.0/students' AS
(a1:chararray,a2:int,a3:int);
DUMP stud;
(raju,2,4)
```

```
(raju,2,4)
(vinay,3,2)
(vinni,3,6)
(vinni,3,6)
```

In this example, all duplicate tuples are removed.

```
rest = DISTINCT stud;
DUMP rest;
raju,2,4)
(vinni,3,6)
```

JOIN

In Pig Latin, you can use the JOIN operator to join one file to another. The JOIN operator begins by demonstrating the keys for every input. There is joining of the two rows if the key are identical. If few records for two inputs do not match, you can drop the particular record.

The various types of joins that can be performed in Pig Latin are:

- Inner join
- Equijoin
- Join of two or more relations based on common field values

The syntax of the JOIN operator is:

```
alias = JOIN alias BY {expression|'('expression [, expression…]')'}
(, alias BY {expression|'('expression [, expression…]')'}…) [USING
'replicated' | 'skewed' | 'merge' | 'merge-sparse']
```

Table 6 defines some common terms of the JOIN operator:

TABLE 6 Common Terms of the JOIN Operator

Term	Description
Alias	Describes the name of a relation
BY	Signifies a key word
Expression	Refers to a field expression Example: X = JOIN A BY fieldA, B BY fieldB, C BY fieldC;
USING	Refers to a key word
'replicated'	Is used to perform replicated joins
'skewed'	Is used to perform skewed joins
'merge'	Is used to perform merge joins
'merge-sparse'	Is used to perform merge-sparse joins

Self Joins

To perform self joins, the same data needs to be loaded using different aliases in Pig. You have to do so to avoid naming conflicts.

The following example illustrates how the same data can be loaded two times using the aliases A and B:

```
grunt> data = load 'mydata';
grunt> data1 = load 'mydata';
grunt> result = join data by $0, data1 by $0;
grunt> explain result;
```

Outer Joins

To perform outer joins in Pig, records that do not have a match on the other side are included with null values filled in for the missing fields. Outer joins are of three types, which are explained as follows:

- A **left outer join** includes records from the left side even when they do not have a match on the right side.
- A **right outer join** includes records from the right side even when they do not have a match on the left side.
- A **full outer join** indicates that records from both sides are taken even when they do not have matches.

QUICK TIP

Outer is a noise word and can be omitted. Unlike some SQL implementations, **full** is not a noise word.
C = join A by x outer, B by u;
will generate a syntax error, not a full outer join.

The following commands depict a sample left join program in Pig:

```
--leftjoin.pig
daily = load 'NYSE_daily' as (exchange, symbol, date, open, high, low, close,
volume, adj_close);
divs = load 'NYSE_dividends' as (exchange, symbol, date, dividends);
jnd = join daily by (symbol, date) left outer, divs by (symbol, date);
```

LIMIT

In Pig, the LIMIT operator enables you to limit the number of results. The syntax of the LIMIT operator is:

```
alias = LIMIT alias n;
```

Table 7 defines some common terms of the LIMIT operator:

TABLE 7
Common Terms of the LIMIT operator

Term	Description
alias	Describes the name of a relation
n	Describes the number of output tuples, which can be either: • A constant (for example, 10) or • A scalar used in an expression (for example, h.add/10)

QUICK TIP

The LIMIT operator allows Pig to avoid processing all tuples in a relation. In most cases, a query that uses the LIMIT operator will run more efficiently than the one that does not use it.

The following example will give an output of three tuples:

```
a = load '/home/titan/hadoop/pig/pig-0.12.0/students' AS
(a1:int,a2:int,a3:int);
DUMP a;
(2,2,3)
(3,2,4)
(7,3,5)
(3,2,3)
Z = LIMIT a 3;
DUMP Z;
(2,2,3)
(3,2,3)
```

SAMPLE

In Pig, the SAMPLE operator can be used to select a random data sample by stating a sample size. The SAMPLE operator expresses the returned percentage of rows in double values. For instance, if the operator returns 0.2, it indicates 20%. The SAMPLE operator is a probabilistic operator, which means that there is no guarantee that the exact same number of tuples will be returned for a particular sample size each time the operator is used. The syntax of the SAMPLE operator is:

SAMPLE alias size

Table 8 presents some common terms of the SAMPLE operator:

TABLE 8
Common Terms of the SAMPLE Operator

Term	Description
alias	Describes the name of a relation
n	Describes the sample size, which can be either: • A constant that ranges from 0 to 1 (for example, enter 0.2 for 20%) or • A scalar used in an expression

In the following example, relation X will contain 1% of the data in relation A:

```
sum= LOAD 'data' AS (f1:int,f2:int,f3:int);
X = SAMPLE sum 0.01;
```

Knowledge Check—2

1. In Pig, which of the following join will you use to get records from the left side even when they do not have a match on the right side?
   a. Left outer join
   b. Right outer join
   c. Full outer join
   d. Normal join

COGROUP

This groups two or more datasets by a column and join based on the same column. Note that COGROUP on multiple datasets results in a record with a key and one bag per dataset.

UNION

This is used to concatenate two datasets together.

RANK

This is used to rank each tuple in a relation.

The implementation of these operators will be demonstrated in detail in the Class.

Exam Checks

Post completion of this class, you are expected to:

- ❍ Demonstrate your ability to use relational operators in Pig

Cheat Sheet

- Relational operators are used to transform data by filtering, grouping, sorting, and joining.
- The FOREACH operator iterates through each record.
- The FILTER operator filters records based on a criteria.
- The GROUP operator groups the data in one or more relations.
- The OrDER BY operator sorts relation based on one or more fields.
- The DISTINCT operator removes duplicate fields from the records.
- The JOIN operator joins one file to another.
- The LIMIT operator limits the number of results.
- The SAMPLE operator selects a random data sample by stating a sample size.
- The COGROUP operator groups two or more datasets by a column and join based on the same column.
- The UNION operator concatenates two datasets together.
- The RANK operator ranks each tuple in a relation.