Table of Contents

Introduction to Pig

Slides

Installing pig

Follow the following steps to install pig on your machine.

Step 1: Download tar.gz from apache site

```
wget http://apache.bytenet.in/pig/pig-0.14.0/\
pig-0.14.0.tar.gz
```

Step 2: Extract tar.gz

```
tar -zxvf pig-0.14.0.tar.gz
```

Step 3 : Start pig

```
bin/pig
```

Pig examples

Loading data

Creating a relationship from data

API

- Load

Code

```
inputData = LOAD '/input/sales.csv' using PigStorage(',')
as (transactionId:chararray,
customerId:chararray, itemId:chararray, amount:Double);

dump inputData;
```

Itemwise count

Problem statement

Find item wise count from the sales data.

API

- multiple for each

Problem description

For any retailer, finding item wise sales is very important. It determines which item is selling well and which is not.

code

```
itemRelation = foreach inputData generate itemId , 1;

groupRelation = group itemRelation by itemId;

countRelation = foreach groupRelation generate
group,COUNT($1);

dump countRelation;
```

Find a specific item

Filter all instances of a given item specified by user.

Problem description

In any analytics, search plays very important role. In this use case, user want to specify an item code and want to get all the instances of that item sale.

API used

- filter

code

```
filteredData = filter inputData by itemId=='1';
dump filteredData;
```

Error Handling

Seperate error records from normal records

Problem description

In big data, malformed data is normal. It is very important to seperate error data from actual data.

API used

- split

code

```
rawData = LOAD '/input/malformedSales.csv'
using PigStorage(',') as (transactionId:chararray,
customerId:chararray, itemId:chararray,
amount:Double);

split rawData into malformed if (amount is null) ,
normalData if (amount is not null);

dump malformed;

dump normalData;
```

Discount

Item wise

Problem statement

Apply optional sale wise discount.

Problem description.

Having discount in retail is very common. But these discount are not always present. So we need to make our code flexible enough to enable and disable the discount at will.

API

- Multi foreach

Code

```
customerAmount = foreach inputData generate customerId ,
amount;

discountAmount =  foreach customerAmount generate
customerId, amount-amount*.05;

groupRelation = group discountAmount by customerId;

customerAmountGroup = foreach groupRelation generate
group,discountAmount.$1;

sumByCustomer = foreach customerAmountGroup generate
group,SUM($1);

dump sumByCustomer;
```

Sum wise

Sum based

Problem statement

Apply discount if total amount is above some threshold.

Problem description

Sometimes, rather than giving item wise discount, we may want to give discount depending on total amount.

code

```
customerAmount = foreach inputData generate
customerId , amount;

groupRelation = group customerAmount by customerId;

customerAmountGroup = foreach groupRelation generate
group,customerAmount.$1;

sumByCustomer = foreach customerAmountGroup generate
group,SUM($1) as totalSum;

discountedCustomers =  filter sumByCustomer by
totalSum > 500.0;

dump discountedCustomers;
```

Joins

Problem statement

Combine sales data with customer master data.

Problem description

Whenever we print bill or do any customer facing communication, we need to specify the customer information along his/her purchase info. But sales data only contains id's not names. So we need to join sales.csv with customers.csv to get the names and sales information in place.

Normal

code

```
customerData = LOAD '/input/customers.csv' using
PigStorage(',') as (customerId:chararray,
customerName:chararray);

joinedData = join inputData by customerId, customerData
by customerId;

dump joinedData;
```

Replicated

code

```
joinedData = join inputData by customerId,
customerData by customerId using 'replicated';
dump joinedData;
```

Note that no reducer ran;

Merge joins

code

```
sortedSalesData = order inputData by customerId;
sortedCustomerData = order customerData by customerId;
mergeJoin = join sortedSalesData by customerId,
sortedCustomerData by customerId using 'merge';
dump mergeJoin;
```

Store

Text format

code

```
filteredData = filter inputData by itemId=='1';

store filteredData into '/filterOutput';
```

Binary format

code

```
filteredData = filter inputData by itemId=='1';
store filteredData into '/filterBinaryOutput'
using BinStorage();
```

Testing and Debugging commands

Describe command

Used to review the schema of a particular alias.

Code

```
DESCRIBE inputData;
```

Illustrate command

Use it to review how data is transformed through a sequence of Pig Latin statements.

Code

```
ILLUSTRATE inputData;
```

Explain command

Displays execution plans like logical plan , physical plan and Map Reduce plan

Code

```
EXPLAIN inputData;
```

Storage

StoreFunc is used for writing data in sequencefileoutput format.

The following are the steps to use serde.

Add jar

```
register 'pig.jar';
```

Use customer relation for storing into sequence file

```
store customerData into 'pigsequence.seq' using
com.madhukaraphatak.hadooptraining.pig.store.SaleStoreFunc();
```

UDF

Upper

Adding jar

```
register pig-1.0-SNAPSHOT.jar;
```

Using UDF

```
customerData = LOAD '/input/customers.csv'
using PigStorage(',') as (customerId:chararray,
customerName:chararray);

customerUpper = foreach customerData generate
com.madhukaraphatak.hadooptraining.pig.udf.Upper
(customerName);
```

Count

Adding jar

```
register pig-1.0-SNAPSHOT.jar;
```

Using UDF

```
inputData = LOAD '/input/sales.csv' using PigStorage(',')
as (transactionId:chararray, customerId:chararray,
itemId:chararray, amount:Double);

itemRelation = foreach inputData generate itemId , 1;

groupRelation = group itemRelation by itemId;

countRelation = foreach groupRelation generate
group,com.madhukaraphatak.hadooptraining.pig.udf.Count
($1);

dump countRelation;
```

```
register /home/hadoop/pig-1.0-SNAPSHOT.jar;
```