Name : Mohit Limbachiya

Roll no: D-26

Day 7

1. **HDFS Commands: -ls, -ls -R, -mkdir, -put, -get**

1) Create a file "Sample" in a local file system and export it to the HDFS File System.

```
hdfs dfs -mkdir /Sample
```

2) Write the HDFS command for copying a "Sample" file from HDFS to local File System.

```
hdfs dfs -put -f E:/Sample.txt \Sample
```

3) Write HDFS commands for creating "Test" directory in HDFS and then removing that directory

```
hdfs dfs -mkdir /Test
hdfs dfs -rmdir /Test
```

4) Write HDFS command to display complete list of directories and files of HDFS.

```
hdfs dfs -ls -R /
```

5) Write HDFS command for displaying the contents of "Sample" text file in HDFS on screen

```
hdfs dfs -cat /Sample/Sample.txt
```

Output:-

```
M0HIT, 22, MCA
```

6) Write HDFS command for copying an existing "Sample" file in a "Test" HDFS directory to some another HDFS directory.

```
hdfs dfs -cp /Sample/Sample.txt /Test
```

2. **Practice HDFS command**

1. Execute the HDFS command for getting the list of complete directories and files of HDFS.

```
hdfs dfs -ls -R /
```

2. Execute the HDFS command for displaying the contents of some Xyz. text file in HDFS on screen.

```
hdfs dfs -cat /Test/Xyz.txt
```

output:-

```
 This is a sample test file
```

3. Execute the HDFS command for copying an existing sample file in a given HDFS directory to some another HDFS directory

```
hdfs dfs -cp /Test1 /Test2
```

Day 8

HDFS Commands: -copyFromLocal, -copyToLocal, -cat, -cp, -rm-r

To get the list of all the files in the HDFS root directory

1. Help

```
hdfs dfs -help
```

Output:-

```
shows help section........
```

2. Write a command to Listing all the files in HDFS.

```
hdfs dfs -ls /
```

3. Write a command to copy of a file “Abc.txt” from Local file System to Hadoop FS.

```
hdfs dfs -copyFromLocal E:/Abc.txt \Abc
```

Practice HDFS command

1. Taking any data/file of your choice, execute the HDFS command for copying a given sample file in local file system to HDFS.

```
hdfs dfs -copyFromLocal E:/Abc.txt \Abc
```

2. Taking any data/file of your choice, execute the HDFS command for copying a given sample file in HDFS to local File System.

```
hdfs dfs -copyToLocal \Abc E:/Abc.txt
```

3. Execute the HDFS commands for creating some sample directory in HDFS and then removing that directory

```
hdfs dfs -mkdir /sample (to Create the directory)

hdfs dfs -rmdir /sample  (to Remove the directory)
```

Day 9

Working with Pig Operators/Functions (LOAD, DUMP, FOREACH, GROUP,DISTINCT,LIMIT, ORDER BY)

Write a pig script to load and store "Student data". (Student file contain Roll no, Name, Marksand GPA)

```
001,Rajiv Reddy,230,4.5
002,siddarth Battacharya,560,2.3
003,Rajesh,Khanna,340,7.8
004,Preethi,Agarwal,356,4.5
005,Trupthi,Mohanthy,290,3.4
006,Archana,Mishra,250,4.3
C:\>cd hadoop-2.9.2 \sbin
C:\hadoop-2.9.2\sbin>start-all
```

//load file in pig_data folder

```
C:\hadoop-2.9.2\sbin>hdfs dfs -copyFromLocal d:\pig\student1.txt /pig_data
```

display:

```
C:\hadoop-2.9.2\sbin>hdfs dfs -cat /pig_data/student1.txt
'C:\Program' is not recognized as an internal or external command,
operable program or batch file.
001,Rajiv Reddy,230,4.5
002,siddarth Battacharya,560,2.3
003,Rajesh,Khanna,340,7.8
004,Preethi,Agarwal,356,4.5
005,Trupthi,Mohanthy,290,3.4
006,Archana,Mishra,250,4.3
```

open pig then open new cmd then write pig command in

command prompt

```
students = LOAD 'hdfs://localhost:9000/pig_data/student2.txt' USING PigStorage (',') as
(rollno:int,name:chararray,marks:int,cgpa:float); grunt>
dump students
```

a. Filter all the students who are having GPA>5 cgpa = FILTER students BY (float)cgpa >

5;dump cgpa;

```
(3,Rajesh Khanna,340,7.8)
```

b. Display the name of all Students in Uppercase.

```
 uname = FOREACH students GENERATE (rollno,name),UPPER(name);
dump uname;  uname = FOREACH students GENERATE
(rollno,name),LOWER(name);
```

c. **Remove duplicates tuple of Student list.**

```
dist_data = DISTINCT students; dump
dist_data;
(1,Rajiv Reddy,230,4.5)
(2,siddarth Battacharya,560,2.3)
(3,Rajesh Khanna,340,7.8)
(4,Preethi Agarwal,356,4.5)
(5,Trupthi Mohanthy,290,3.4)
(6,Archana Mishra,250,4.3)
```

d. **Display first three tuples from “student” relation.**

```
limit_data = LIMIT students 3; dump
limit_data;
```

----

e. Display the names of students in ascending order asce = ORDER students BY name ASC;dump asce; last 3 record show

```
asce = ORDER students  BY name ASC;
last = LIMIT asce 3;

dump last;


(6,Archana Mishra,250,4.3)
(4,Preethi Agarwal,356,4.5)
(3,Rajesh Khanna,340,7.8)
```

----

csv file load

```
C:\hadoop-2.9.2\sbin>hdfs dfs -copyFromLocal d:\pig\movie.csv /pig_data


C:\hadoop-2.9.2\sbin>hdfs dfs -cat /pig_data/movie.csv
'C:\Program' is not recognized as an internal or external command, operable
program or batch file.
1,DDLJ,1986,3.2,7560
2,Xyz,1985,3.8,6300
3,ABC,1988,4.1,7802
4,PQR,1993,3.7,6022 5,AAA,1991,3.4,5420
6,ZZY,2004,3.9,4904
7,De danadan,1987,3.4,5623
8,GCET,1987,3.4,7563
9,PPP,1990,3.2,6244
10,PQQQ,2004,3.1,69565
```

```
grunt> movie = LOAD 'hdfs://localhost:9000/pig_data/movie.csv' USING PigStorage (',') as
(id:int,name:chararray,year:int,rating:float,duration:int);
grunt> dump movie;
```

----

1. Filter movie whose rating is higher than 3.5

```
rating = FILTER movie BY (float)rating > 3.5;

dump rating;
```

```
(2,Xyz,1985,3.8,6300)

(3,ABC,1988,4.1,7802)

(4,PQR,1993,3.7,6022)

(6,ZZY,2004,3.9,4904)
```

----

2. Store the results data from pig into new name my_movies.

```
store movies into 'my_movie';
```

```
cat my_movie;

grunt> cat movies;

1      DDLJ    1986    3.2     7560

2      Xyz     1985    3.8     6300

3      ABC     1988    4.1     7802

4      PQR     1993    3.7     6022 5      AAA     1991    3.4     5420

6       ZZY     2004    3.9     4904

7       De danadan      1987    3.4     5623

8       GCET    1987    3.4     7563

9       PPP     1990    3.2     6244

10      PQQQ    2004    3.1     69565
```

----

3. Display all movie information the result.

```
grunt> cat movies;

1      DDLJ    1986    3.2     7560

2      Xyz     1985    3.8     6300

3      ABC     1988    4.1     7802

4      PQR     1993    3.7     6022 5      AAA     1991    3.4     5420

6       ZZY     2004    3.9     4904

7       De danadan      1987    3.4     5623

8       GCET    1987    3.4     7563

9       PPP     1990    3.2     6244

10      PQQQ    2004    3.1     69565
```

```
grunt> describe movie; movie: {id: int,name: chararray,year:

int,rating: float,duration: int}
```

```
grunt> illustrate movie;


-------------------------------------------------------------------------------------------------------------------

| movie     | id:int     | name:chararray     | year:int     | rating:float     | duration:int     |

-------------------------------------------------------------------------------------------------------------------

|          | 5         | AAA            | 1991       | 3.4           | 5420          | ---------------------------------
-----------------------------------------------------------------------------
```

```
grunt> explain movie;

--------------------------------------------------------------------------------------
```

4. List the movies that were released between 1950 and 1960.

```
year = FILTER movie by year > 1990 and year < 2004;

dump year;
(4,PQR,1993,3.7,6022) (5,AAA,1991,3.4,5420)
```

----

5. List the movies that start with the Alphabet D.

```
z = FILTER movie BY name matches 'D.*'; dump
z;

(1,DDLJ,1986,3.2,7560)
(7,De danadan,1987,3.4,5623)
```

----

6. List the movies that have duration greater than 2 hours.

```
dur  = FILTER movie by duration > 7200;

dump dur;
(1,DDLJ,1986,3.2,7560)
(3,ABC,1988,4.1,7802)
(8,GCET,1987,3.4,7563)
(10,PQQQ,2004,3.1,69565)
```

----

7. List the movie names its duration in minutes.

```
dur = FOREACH movie GENERATE name, (double)(duration/60);
```

```
dump dur;
(DDLJ,126.0)
(Xyz,105.0)
(ABC,130.0)
(PQR,100.0)
(AAA,90.0)
(ZZY,81.0)
(De danadan,93.0)
(GCET,126.0)
(PPP,104.0)
(PQQQ,1159.0)
```

or

```
dur = FOREACH movie GENERATE name, (double)(duration/60),(duration/3600);
dump dur; (DDLJ,2.0)
(Xyz,1.0)
(ABC,2.0)
(PQR,1.0) (AAA,1.0)
(ZZY,1.0)
```

```
(De danadan,1.0)

(GCET,2.0)

(PPP,1.0)

(PQQ,19.0)
```

Group Statement in PIG.

8. List the years and the number of movies released each year.

```
year = GROUP movie BY year;



dump year;
(1985,{(2,Xyz,1985,3.8,6300)})
(1986,{(1,DDLJ,1986,3.2,7560)})
(1987,{(8,GCET,1987,3.4,7563),(7,De danadan,1987,3.4,5623)})
(1988,{(3,ABC,1988,4.1,7802)})
(1990,{(9,PPP,1990,3.2,6244)})
(1991,{(5,AAA,1991,3.4,5420)}) (1993,{(4,PQR,1993,3.7,6022)})
(2004,{(10,PQQQ,2004,3.1,69565),(6,ZZY,2004,3.9,4904)})

cout_year = FOREACH year GENERATE group,COUNT(movie);

dump cout_year;
(1985,1)
```

```
(1986,1)
(1987,2)
(1988,1)
(1990,1)
(1991,1)
(1993,1)
(2004,2)
```

Order by in PIG Statement.

9. List all the movies in the ascending order of year.

```
order_by = ORDER movie BY name ASC;

dump order_by;
(5,AAA,1991,3.4,5420)
(3,ABC,1988,4.1,7802)
(1,DDLJ,1986,3.2,7560)
(7,De danadan,1987,3.4,5623)
(8,GCET,1987,3.4,7563)
(9,PPP,1990,3.2,6244)
(10,PQQQ,2004,3.1,69565)
(4,PQR,1993,3.7,6022)
(2,Xyz,1985,3.8,6300)
(6,ZZY,2004,3.9,4904)
```

```
order_by = ORDER movie BY year ASC;

dump order_by;
(2,Xyz,1985,3.8,6300)
(1,DDLJ,1986,3.2,7560)
(8,GCET,1987,3.4,7563)
(7,De danadan,1987,3.4,5623)
(3,ABC,1988,4.1,7802)
(9,PPP,1990,3.2,6244)
(5,AAA,1991,3.4,5420) (4,PQR,1993,3.7,6022)
(10,PQQQ,2004,3.1,69565)
(6,ZZY,2004,3.9,4904)
```

----

10. List all the movies in the descending order of year.

```
order_by = ORDER movie BY year DESC;

dump order_by;
(10,PQQQ,2004,3.1,69565)
(6,ZZY,2004,3.9,4904)
(4,PQR,1993,3.7,6022) (5,AAA,1991,3.4,5420)
(9,PPP,1990,3.2,6244)
(3,ABC,1988,4.1,7802)
(8,GCET,1987,3.4,7563)
(7,De danadan,1987,3.4,5623)
```

```
(1,DDLJ,1986,3.2,7560)
(2,Xyz,1985,3.8,6300)
```

Limit operator in pig.

11. Display Top 5 movies.

```
order_by = ORDER movie BY year ASC;

dump order_by;
(2,Xyz,1985,3.8,6300)
(1,DDLJ,1986,3.2,7560)
(8,GCET,1987,3.4,7563)
(7,De danadan,1987,3.4,5623)
(3,ABC,1988,4.1,7802)
(9,PPP,1990,3.2,6244)
(5,AAA,1991,3.4,5420) (4,PQR,1993,3.7,6022)
(10,PQQQ,2004,3.1,69565)
(6,ZZY,2004,3.9,4904)

top5 = LIMIT order_by 5;

dump top5;
(2,Xyz,1985,3.8,6300)
(1,DDLJ,1986,3.2,7560)
(7,De danadan,1987,3.4,5623)
```

```
(8,GCET,1987,3.4,7563)
(3,ABC,1988,4.1,7802)
```

```
grunt> describe movie; movie: {id: int,name: chararray,year:
int,rating: float,duration: int}
```

```
grunt> illustrate movie;

--------------------------------------------------------------------------------------------
| movie    | id:int    | name:chararray    | year:int    | rating:float    | duration:int    |
--------------------------------------------------------------------------------------------
|          | 5         | AAA               | 1991        | 3.4             | 5420            |
--------------------------------------------------------------------------------------------
```

```
grunt> explain movie;
```

3. <u>Load the file menu.csv (Category, Name, Price) and write one Pig script</u>

```
C:\hadoop-2.9.2\sbin>hdfs dfs -copyFromLocal d:\pig\menu.csv /pig_data
'C:\Program' is not recognized as an internal or external command, operable
program or batch file.
```

```
C:\hadoop-2.9.2\sbin>hdfs dfs -cat /pig_data/menu.csv
'C:\Program' is not recognized as an internal or external command, operable
program or batch file.
SouthIndain,PannerDosa,15
```

```
SouthIndain,Idli,10

Chinese,Manchurian,200

Chinese,PannerNoodles,40

Continental,Pizza,70

Gujarati,Thali,500

Thai,Kari,30

NorthIndian,PannerDosa,50
```

```
grunt> menu = LOAD 'hdfs://localhost:9000/pig_data/menu.csv' USING PigStorage(',') as
(category:chararray,name:chararray,price:int);

grunt> dump menu;
```

a. Which meals cost more than 30.00?

```
grunt> price = FILTER menu BY price > 30; grunt>

dump price;
```

```
(Chinese,Manchurian,200)

(Chinese,PannerNoodles,40)

(Continental,Pizza,70)

(Gujarati,Thali,500)

(NorthIndian,PannerDosa,50)
```

b. Which meals contain the word “Panner”?

```
grunt> panner = FILTER menu by name matches '.*Panner.*'; grunt>

dump panner;
```

```
(SouthIndain,PannerDosa,15) (Chinese,PannerNoodles,40)
```

```
(NorthIndian,PannerDosa,50)
```

c. Which are the 10 most expensive meals?

```
grunt> meals = ORDER menu BY price DESC; grunt>
dump meals;

grunt> top_10_exp =  LIMIT meals 10; grunt>
dump top_10_exp; (Gujarati,Thali,500)
(Chinese,Manchurian,200)
(Continental,Pizza,70)
(NorthIndian,PannerDosa,50)
(Chinese,PannerNoodles,40)
(Thai,Kari,30)
(SouthIndain,PannerDosa,15)
(SouthIndain,Idli,10)
```

d. For every day, what’s the average price for a meal?

```
//optional meal = LOAD 'hdfs://localhost:9000//pig_data/menu.csv' USING PigStorage(',') as
(category:chararray,name:chararray,price:int);

grunt> menu_group = GROUP menu ALL; grunt>
dump menu_group;
```

```
grunt> avg_price = foreach menu_group Generate (menu.category,menu.name,menu.price),
AVG(menu.price); grunt> dump avg_price;
```

```
(({(NorthIndian),(Thai),(Gujarati),(Continental),(Chinese),(Chinese),(SouthIndain),(SouthIndain)},
{(PannerDosa),(Kari),(Thali),(Pizza),(PannerNoodles),(Manchurian),(Idli),(PannerDosa)}),114.375
)
```

e. For every day, what's the most expensive meal

```
grunt> exp = order meal by price desc;
grunt> most_exp = LIMIT exp 1; grunt>
dump most_exp;
(Gujarati,Thali,500)
```

or

```
grunt> menu_group = GROUP menu ALL; grunt>
dump menu_group;
```

```
grunt> max_price = foreach menu_group Generate (menu.category,menu.name,menu.price),
MAX(menu.price); grunt>  dump max_price; hdfs dfsadmin -safemode leave // when no create
direcatory in cmd then use this command
```

Day 10

```
C:\hadoop-2.9.2\sbin>hdfs dfs -copyFromLocal  d:\day10\customers.txt /pig_data
'C:\Program' is not recognized as an internal or external command, operable
program or batch file.
```

```
C:\hadoop-2.9.2\sbin>hdfs dfs -copyFromLocal d:\day10\orders.txt /pig_data
'C:\Program' is not recognized as an internal or external command, operable
program or batch file.
```

load in pig

```
cust = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING PigStorage(',') as
(c_id:int,name:chararray,age:int,city:chararray,amount:int); dump
cust;

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

```
grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING PigStorage(',') as
(id:int,date:chararray,c_id:int,amount:int); grunt>
dump orders;
(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)
```

UNION two table merge customer,orders;

Union:

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

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

Split:

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.

```
SPLIT customer into Below_23 if age<25, Above_25 if (age>=25);
 dumpBelow_23;
(3,kaushik,23,Kota,2000)
(6,Komal,22,MP,4500)
(7,Muffy,24,Indore,10000)

SPLIT customer into Below_23 if age<23,bet22_25 if (age>22 and age<25);
```

```
 dump Below_23;
(6,Komal,22,MP,4500)

dump bet22_25;
(3,kaushik,23,Kota,2000)
(7,Muffy,24,Indore,10000)
```

salary split

```
SPLIT customer into below_2000 if amount <2000, above_2000 if (amount>=2000);

dump below_2000;

(2,Khilan,25,Delhi,1500)

SPLIT customer into below_2000 if amount<2000,above_2000 if(amount>=2000 and
 amount<=9000); dump below_2000;

(2,Khilan,25,Delhi,1500)

dump above_2000;
(1,Ramesh,32,Ahmedabad,2000)
(3,kaushik,23,Kota,2000)
(4,Chaitali,25,Mumbai,6500)
(5,Hardik,27,Bhopal,8500)
(6,Komal,22,MP,4500)
```

Filter:

Use the Filter operator to get the details of the students who belong to the city Chennai

```
cust_city = FILTER customer BY(chararray)city == 'Ahmedabad';  dump cust_city;
```

```
(1,Ramesh,32,Ahmedabad,2000)
```

Distinct:

remove the redundant (duplicate) tuples from the relation named student_detail

```
grunt> dist_deta = DISTINCT cust_group; grunt>
dump dist_deta;
```

```
(all,{(7,Muffy,24,Indore,10000),(6,Komal,22,MP,4500),(5,Hardik,27,Bhopal,8500),(4,Chaitali,25,
Mumbai,6500),(3,kaushik,23,Kota,2000),(2,Khilan,25,Delhi,1500),(1,Ramesh,32,Ahmedabad,20
00)})
```

FOREACH operator:

get the id, age, and city values of each student from the relation student_details and store it into another relation named student_data

```
cust_data = FOREACH customer GENERATE id , age , city;

store cust_data into 'customer_data';

cat customer_data;
```

```
1       32      Ahmedabad
2       25      Delhi
3       23      Kota
4       25      Mumbai
5       27      Bhopal
6       22      MP
7       24      Indore
```

aggerate function throw:function (AVG(),MIN(),MAX(),COUNT(),SUM())

```
cust_group = Group Cust_det ALL;
```

```
grunt>  avg_age =  FOREACH cust_group GENERATE (Cust_det.name,Cust_det.age),
AVG(Cust_det.age); grunt> dump avg_age;
(({(Muffy),(Komal),(Hardik),(Chaitali),(kaushik),(Khilan),(Ramesh)},{(24),(22),(27),(25),(23),(25),(3
2)}),25.428571428571427)
```

----

join

SELF JOIN

```
grunt> cust_self = JOIN cust BY c_id, orders BY c_id; grunt>
dump cust_self;
(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)
```

inner join

```
grunt> cust_order = JOIN cust BY c_id, orders BY c_id; grunt>
dump cust_order;
(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)
```

left outer join outer_left = JOIN orders BY c_id LEFT
OUTER, cust BY c_id; dump outer_left;

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

or

```
grunt> outer_left = JOIN cust BY c_id LEFT OUTER, orders BY c_id; grunt>
dump 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

```
outer_right = JOIN orders BY c_id RIGHT, cust BY c_id; dump
outer_right;

(,,,,1,Ramesh,32,Ahmedabad,2000)
(101,2009-11-20 00:00:00,2,1560,2,Khilan,25,Delhi,1500)
(100,2009-10-08 00:00:00,3,1500,3,kaushik,23,Kota,2000)
(102,2009-10-08 00:00:00,3,3000,3,kaushik,23,Kota,2000)
(103,2008-05-20 00:00:00,4,2060,4,Chaitali,25,Mumbai,6500)
(,,,,5,Hardik,27,Bhopal,8500)
(,,,,6,Komal,22,MP,4500) (,,,,7,Muffy,24,Indore,10000) or
grunt> outer_right = JOIN cust BY c_id RIGHT, orders BY c_id;
grunt> dump 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 join

```
grunt> outer_full = JOIN cust BY c_id FULL OUTER, orders BY c_id; grunt>
dump 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,,,,)
```

```
TOKENIZE grunt> cust_tok = foreach cust Generate
TOKENIZE(name); grunt> dump cust_tok;
({(Ramesh)})
({(Khilan)})
({(kaushik)}) ({(Chaitali)})
({(Hardik)})
({(Komal)})
({(Muffy)})
```

TOBAG:

```
grunt> tobag = FOREACH cust GENERATE TOBAG (c_id,name,age); grunt>
dump tobag;
({(1),(Ramesh),(32)})
({(2),(Khilan),(25)})
({(3),(kaushik),(23)}) ({(4),(Chaitali),(25)})
({(5),(Hardik),(27)})
```

```
({(6),(Komal),(22)})
({(7),(Muffy),(24)})
```

```
grunt> cust_tuple = FOREACH cust GENERATE TOTUPLE (c_id,name,age); grunt>
dump cust_tuple;.
```

```
((1,Ramesh,32))
((2,Khilan,25))
((3,kaushik,23)) ((4,Chaitali,25))
((5,Hardik,27))
((6,Komal,22))
((7,Muffy,24))
```

TOMAP

```
grunt> cust_map = FOREACH cust GENERATE TOMAP (name,age); grunt>
dump cust_map;
```

```
([Ramesh#32])
([Khilan#25])
([kaushik#23])
 ([Chaitali#25])
([Hardik#27])
([Komal#22])
([Muffy#24])
```

TOP

```
grunt> cust_group = GROUP cust BY age; grunt>
dump cust_group;
```

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

```
cust_top = FOREACH cust_group{
        top=Top(2, 0, cust);
        GENERATE top;
};
```

Day 11

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

```
ord = order emp_data by age;
(12,Kelly,22,Chennai)
(7,Robert,22,newyork )
(6,Maggy,22,Chennai )
(1,Robin,22,newyork )
(8,Syam,23,Kolkata )
(5,David,23,Bhuwaneshwar )
(3,Maya,23,Tokyo )
(2,BOB,23,Kolkata )
```

```
(11,Stacy,25,Bhuwaneshwar )

(10,Saran,25,London )

(9,Mary,25,Tokyo )

(4,Sara,25,London )
```

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

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

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 )})
```

```
data_top = FOREACH emp_group {
   top = TOP(1, 0, emp_data);
   GENERATE top;
}
({(12,Kelly,22,Chennai)})
({(8,Syam,23,Kolkata )})
({(11,Stacy,25,Bhuwaneshwar )})
```

```
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 )})
```

String Functions & Description

```
------------------------------------------------
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
-------------------
```

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

1 ENDSWITH(string, testAgainst)

To verify whether a given string ends with a particular substring.

emp_endswith = FOREACH emp_data GENERATE (id,name),ENDSWITH (name, 'n');

----

2 STARTSWITH(string, substring)

Accepts two string parameters and verifies whether the first string starts with the second.

startswith_data = FOREACH emp_data GENERATE (id,name), STARTSWITH (name,'Ro');

----

3 SUBSTRING(string, startIndex, stopIndex)

Returns a substring from a given string. substring_data = FOREACH emp_data

GENERATE (id,name), SUBSTRING (name, 0, 2);

----

4 **EqualsIgnoreCase(string1, string2)**

To compare two stings ignoring the case.

equals_data = FOREACH emp_data GENERATE (id,name), EqualsIgnoreCase(name, 'Robin');

----

5 INDEXOF(string, ‘character’, startIndex)

Returns the first occurrence of a character in a string, searching forward from a start index.
indexof_data = FOREACH emp_data GENERATE (id,name), INDEXOF(name, 'r',0);

----

6 LAST_INDEX_OF(expression)

Returns the index of the last occurrence of a character in a string, searching backward from a start index.

last_index_data = FOREACH emp_data GENERATE (id,name), LAST_INDEX_OF(name, 'g');

----

7 LCFIRST(expression)

Converts the first character in a string to lower case.

Lcfirst_data = FOREACH emp_data GENERATE (id,name), LCFIRST(name);

----

8 UCFIRST(expression)

Returns a string with the first character converted to upper case.

ucfirst_data = FOREACH emp_data GENERATE (id,city), UCFIRST(city);

----

9 UPPER(expression)

UPPER(expression) Returns a string converted to upper case.

upper_data = FOREACH emp_data GENERATE (id,name), UPPER(name);

----

10 LOWER(expression)

Converts all characters in a string to lower case.

----

11 REPLACE(string, ‘oldChar’, ‘newChar’);

To replace existing characters in a string with new characters.

replace_data = FOREACH emp_data GENERATE (id,city),REPLACE(city,'Bhuwaneshwar','Bhuw'); --
----

12 STRSPLIT(string, regex, limit)

To split a string around matches of a given regular expression. strsplit_data =

FOREACH emp_data GENERATE (id,name), STRSPLIT (name,'_',2);

----

13 STRSPLITTOBAG(string, regex, limit)

Similar to the STRSPLIT() function, it splits the string by given delimiter and returns the result in a bag.

```
 strsplittobag_data = FOREACH emp_data GENERATE (id,name), STRSPLITTOBAG (name,'_',2);
```

----

14 TRIM(expression)

Returns a copy of a string with leading and trailing whitespaces removed.

```
trim_data = FOREACH emp_data GENERATE (id,name), TRIM(name);
```

----

15 LTRIM(expression)

Returns a copy of a string with leading whitespaces removed.

```
 ltrim_data = FOREACH emp_data GENERATE (id,name), LTRIM(name);
```

----

16 RTRIM(expression)

Returns a copy of a string with trailing whitespaces removed.

```
rtrim_data = FOREACH emp_data GENERATE (id,name), RTRIM(name);
```

date.txt

```
--------------
001,1989/09/26 09:00:00
002,1980/06/20 10:22:00
003,1990/12/19 03:11:44
--------------
```

```
date_data = LOAD 'hdfs://localhost:9000/emp/date.txt' USING PigStorage(',')
   as (id:int,date:chararray);
```

1 ToDate(milliseconds)

This function returns a date-time object according to the given parameters. The other alternative for this function are ToDate(iosstring), ToDate(userstring, format), ToDate(userstring, format, timezone)

todate_data = foreach date_data generate ToDate(date,'yyyy/MM/dd HH:mm:ss')

as (date_time:DateTime >);

----

2 CurrentTime()

returns the date-time object of the current time.

currenttime_data = foreach todate_data generate CurrentTime();

----

3 GetDay(datetime)

Returns the day of a month from the date-time object. getday_data = foreach

todate_data generate(date_time), GetDay(date_time);

----

4 GetHour(datetime)

Returns the hour of a day from the date-time object.

gethour_data = foreach todate_data generate (date_time), GetHour(date_time);

5 GetMilliSecond(datetime)

Returns the millisecond of a second from the date-time object.

6 GetMinute(datetime)

Returns the minute of an hour from the date-time object.

7 GetMonth(datetime)

Returns the month of a year from the date-time object.

8 GetSecond(datetime)

Returns the second of a minute from the date-time object.

9 GetWeek(datetime)

Returns the week of a year from the date-time object.

10 GetWeekYear(datetime)

Returns the week year from the date-time object.

11 GetYear(datetime)

Returns the year from the date-time object.

----

12 AddDuration(datetime, duration)

Returns the result of a date-time object along with the duration object.

Note − The Duration is represented in ISO 8601 standard. According to ISO 8601 standard P is placed at the beginning, while representing the duration and it is called as duration designator. Likewise,

Y is the year designator. We use this after declaring the year.
Example − P1Y represents 1 year.

M is the month designator. We use this after declaring the month.
Example − P1M represents 1 month.

W is the week designator. We use this after declaring the week.
Example − P1W represents 1 week.

D is the day designator. We use this after declaring the day.
Example − P1D represents 1 day.

T is the time designator. We use this before declaring the time.
Example − PT5H represents 5 hours.

H is the hour designator. We use this after declaring the hour.
Example − PT1H represents 1 hour.

M is the minute designator. We use this after declaring the minute.
Example − PT1M represents 1 minute.

S is the second designator. We use this after declaring the second.

Example − PT1S represents 1 second.

```
date_duration = LOAD 'hdfs://localhost:9000/emp/date.txt' USING PigStorage(',') as (id:int,
date:chararray, duration:chararray)



Add_duration_data = foreach date_duration generate(date,duration),

   AddDuration(ToDate(date,'yyyy/MM/dd HH:mm:ss'), duration);
```

---

1 SubtractDuration(datetime, duration)

Subtracts the Duration object from the Date-Time object and returns the result.

2 DaysBetween(datetime1, datetime2)

Returns the number of days between the two date-time objects.

```
doj_dob_data = LOAD 'hdfs://localhost:9000/pig_data/doj_dob.txt' USING PigStorage(',') as
(id:int, dob:chararray, doj:chararray);
```

```
daysbetween_data = foreach doj_dob_data generate DaysBetween(ToDate
(doj,'dd/MM/yyyy HH:mm:ss'),ToDate(dob,'dd/MM/yyyy HH:mm:ss'));
```

---

1 HoursBetween(datetime1, datetime2)

Returns the number of hours between two date-time objects.

2 MilliSecondsBetween(datetime1, datetime2)

Returns the number of milliseconds between two date-time objects.

3 MinutesBetween(datetime1, datetime2)

Returns the number of minutes between two date-time objects.

4 MonthsBetween(datetime1, datetime2)

Returns the number of months between two date-time objects.

5 SecondsBetween(datetime1, datetime2)

Returns the number of seconds between two date-time objects.

6 WeeksBetween(datetime1, datetime2)

Returns the number of weeks between two date-time objects.

7 YearsBetween(datetime1, datetime2)

Returns the number of years between two date-time objects.