EXPERIMENT - 5

MAPREDUCE PROGRAM TO FIND THE GRADES OF STUDENTS

*AIM: To develop a mapReduce program to find the grades of students

*PROGRAM:

```
import java.util.Scanner;
public class JavaExample
 {
    public static void main (String args[])
       int marks[]= new int[6];
       int i;
       float total=0, avg;
       Scanner scanner = new Scanner (System.in);
       for(i=0; i<6; i++){
          System.out.print("Enter Marks of subject"+
                           (i+1)+":");
          marks[i] = scanner.nextInt();
          total = total + marks[i];
       }
       Scanner.close();
    // here avg = total/6;
       System.out.print(" The student Grade is:");
```

```
if (avg >= 80)
  {
    System.out.print("A");
  }
else if (avg >= 60 && avg < 80)
  {
    System.out.print("B");
  }
else if (avg >= 40 && avg < 60)
  {
    System.out.print("C");
  }
else
  {
    System.out.print("D");
  }
}
}
```

OUTPUT:

Enter marks of Subject 1: 40
Enter marks of Subject 2: 80
Enter marks of Subject 3: 80
Enter marks of Subject 4: 40
Enter marks of Subject 5: 60
Enter marks of Subject 6: 60
The Student Grade is B

EXPERIMENT - 6

*AIM: Retrieving user login credentials from /etc/password using

*PROGRAM:

⇒ step 1: Copy /etc/passwd into pig_programs folder as passwd.txt

⇒ step 2: Create pass_analysis.pig

```
pass = load "passwd.txt" using pigStorage(':') as
(username:chararray, pwd:chararray, userid:int,
groupid:int, userinfo:chararray, root_dir:chararray
shell:chararray);
user_list = foreach pass generate username, pwd;
dump user_list;
```

⇒ step 3: Create pass_analysis1.pig

```
pass = load "passwd.txt" using pigStorage(':') as
(username:chararray, pwd:chararray, userid:int
root_dir:chararray);
grp_by_shell = group pass by shell;
STORE pass_cnt INTO 'pig_output' USING pigStorage(',');
```

Step 4: Pig -x local pass_analysis.pig

Step 5: Pig -x local pass_analysis1.pig

Step 6: Result can be seen through command interface as $hadoop dfs -cat Pig_output/part-r-00000

EXPERIMENT - 7

*AIM: Working with hive

* HIVE PROGRAMS:

=> $Hive:

↳ Create database

Syntax: CREATE DATABASE | SCHEMA [IF NOT EXISTS] <database name>

hive > CREATE DATABASE [IF NOT EXISTS] userdb;

or

hive> CREATE SCHEMA userdb;

↳ Drop Database

Syntax: DROP DATABASE statement DROP [DATABASE | SCHEMA) [IF EXISTS] database_name [RESTRICT | CASCADE];

hive > DROP DATABASE IF EXISTS userdb;

hive > DROP DATABASE IF EXISTS userdb CASCADE;

hive > DROP SCHEMA userdb;

↳ Create Table

Syntax: CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]

[(col_name data_type [COMMENT col_comment],...)]

[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]

hive > CREATE TABLE IF NOT EXISTS employee (eid int, name String, salary salary String, designation String, department String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
STORED AS TEXTFILE;

↳ Create table from existing table
Create table emp select * from employee;

↳ Creating table with buckets:
CREATE TABLE students (sid int, sname String, age INT, dept String, gpa DECIMAL (3, 2))
CLUSTERED BY (dept) INTO 4 BUCKETS
STORED AS ORC;

↳ ALTER TABLE:
Syntax
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec [, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name

Rename To..

hive > ALTER TABLE employee RENAME

Change
hive > ALTER TABLE employee CHANGE name ename String;

Add Column:
hive > ALTER TABLE employee ADD COLUMNS

Replace:
hive ALTER TABLE employee REPLACE COLUMNS (empid Int, ename STRING)

↳ DROP TABLE:
Syntax
DROP TABLE [IF EXISTS] table_name;
hive > DROP TABLE IF EXISTS employee

↳ loading files into Tables:
Syntax:
LOAD DATA [LOCAL] INPATH "filepath" [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1 = val1, partcol2 = val2)]

Insert Data from a table:
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, partcol2 = val2 ...) [IF NOT EXISTS]]

* HIVE - Partitioning:

Hive partitions is a way to organizes tables into partitions by dividing tables into different parts.

1. Creation of Table studentInfo
   Create table studentInfo (sid int, sname String, phoneno BIGINT, dept String
2. loading data into created table all states
3. Creation of partition table.
4. For partition we have to set this property
   set hive.exec.dynamic.partion table

INSERT OVERWRITE TABLE student_part PARTITION(dept)
SELECT * from studentInfo;

a) Adding a partition using Alter Table

Syntax:
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION
partition_spec [LOCATION 'location'] partition_spec
partition_spec (p_column = p_col_value, p_column = p_col_val..)
hive> ALTER TABLE student_part ADD PARTITION

b. DROPPING A PARTITION:

Syntax:

ALTER TABLE table_name DROP [IF EXISTS] PARTITION
partition_spec, PARTITION partition_spec ... ;

Merge two tables

Standard Syntax:

```
MERGE INTO <target table> AS T USING <source expression
/table> AS S
ON <boolean expression1>
WHEN MATCHED [AND <boolean expression2>] THEN
 UPDATE SET <set clause list>
WHEN MATCHED [AND <boolean expression3>] THEN DELETE
```

* HiveQL

Syntax:

```
SELECT [ALL | DISTINCT] select_expr, select_expr...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[limit number];
```

1. Query to retrieve the employee details whose Id is 1205

```
hive> SELECT * FROM employee WHERE Id=1205
```

2. Query to retrieve the employee details whose salary is more than or equal Rs 40000

```
hive> SELECT * FROM employee WHERE salary>=40000;
```

* Simple Queries on employee table.

Id	Name	Salary	Designation	Dept
1201	Gopal	45000	Technical manager	TP
1202	Manisha	45000	proofreader	PR
1203	Kiran	40000	Hr Admin	HR
1204	Kranthi	40000	Op Admin	Admin

3. Query to retrieve employee details whose dept is TP and salary is more than Rs 40000

hive > SELECT * FROM employee WHERE salary>40000 and dept = TP;

* Aggregate functions:

hive> SELECT count(*) FROM employee;
hive > SELECT sum(salary) FROM employee;
hive > SELECT avg(salary) FROM employee;

* Hive - Built - in functions:

Create a dummy table with single row and column & insert a value into it.

Hive > Create table dummy (value String);
Hive > load data local inpath '/home/training/dummydata.csv'

1) round() function
hive> SELECT round(2.6) from dummy;

2) floor() function
hive > SELECT floor(2.6) from dummy;

ADD
50

* VIEW AND INDEXES

→ Creating a View:

↳ Syntax:

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name
  [COMMENT column_comment],...)]
```

Query to creat a view named as emp_30000 with salary > 30000

```
hive> CREATE VIEW emp_30000 AS SELECT * FROM employee
WHERE salary> 30000;
```

→ Dropping a View:

↳ Syntax:

```
DROP VIEW view_name
```

Query to drops a view named as emp_30000:

```
hive> DROP VIEW emp_30000;
```

Creating an Index:

Syntax:

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name,...)
As 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IN TABLE index_table_name]
[
  [ROW FORMAT... ] STORED AS...
  ]
```

Testing:

hive> SELECT avg(Salary) From employee

hive > show formatted index on employee;
hive> CREATE INDEX in_empsalary ON TABLE employee(salary)
hive >SELECT avg(salary) FROM employee;

AS "org.apache.hadoop.hive.ql.index.bitmap.BitmapIndexHandler
hive > ALTER INDEX inbit_empsalary ON employee
REBUILD;
hive > SELECT avg(Salary) fRom employee;

Syntax:

DROP INDEX <index_name> ON <table_name>
The following query drops an index named index_salary

* JOINS:

↳ Syntax:

join_table:
table_reference JOIN table_factor [join_condition]
|table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN
table_reference
Join condition
|table_reference LEFT SEMI JOIN table_reference join
_condition

JOIN on the CUSTOMER and ORDER tables

```
hive> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
FROM CUSTOMERS C JOIN ORDERS ON (C.ID=O.CUSTOMER_ID)
```

LEFT OUTER JOIN between customer and ORDER tables

```
hive> SELECT C.ID, C.NAME, O.AMOUNT
FROM CUSTOMERS C FULL OUTER JOIN ORDERS ON
    (C.ID = O.CUSTOMER_ID);
```

Example

Consider the following table named CUSTOMERS

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Hardik	27	Kota	2000.00
4	Muffy	24	Mumbai	6500.00

⇒ Consider another table ORDERS as follows:

OID	customer-ID	Amount
102	3	3000
100	3	1500
101	2	1560

EXPERIMENT - 8

*AIM: Hive UDF (User Defined functions)

↳ HIVE UDF PROGRAMS:

Steps to write a UDF function in Hive

1. Create a java class for the user defined function which extends org.apache.hadoop.hive.ql.exec.UDF and implements more than one evaluate()
2. Package your java class into a JAR File
3. Go to Hive CLI, add your JAR and verify your JAR's
4. Use it in Hive SQL

↳ Simple UDF for "trim_like" function called "strip"

1. Open Eclipse and create a new project
2. Create a new package with name stripword
3. Create a new class file with name strip
   - ↳ Hadoop/Hadoop-core.jar
   - ↳ Hadoop/lib/common-cli-1.2.jar

*PROGRAM:

```
Package stripword;

import org.apache.commons.lang.StringUtils;
```

```
impmt org.apache.hadoop.hive.ql.exec.UDF;
impmt org.apache.hadoop.io.Text;

public class Ship extends UDF {
    privale Text result = new Text();
    public Text evaluate (Text str, String shipChars) {
        if (str == NULL){
        return null;
        }
    result.set (StringUtils.ship(str.toString(), ShipChars));
    return result;
    }

public Text evaluate (Text str) {
    if (str == null) {
        return null;
        }
    result.set(StringUtils.ship (str.toString()));
    return result;
    }
}
```

Verify JAR is in Hive CLI classpath

```
hive > list jars:
/usr/lib/hive/lib/hive-contrib.jar
/home/training/HiveUDFs/shippingword.jar
```

2. Convert celcius to fahrenheit

```
Package udfpack;
import org.apache.hadoop.hive.ql.exec.UDF;

    public class ConvertToCelcius extends UDF {
        public double evaluate(double value) {
            return (value - 32) / 1.8;
        }
    }
```

```
hive> add jar /home/training/HiveUDFs/convtocel.jar
hive > SELECT ConvertToCelcius(100) from dummy;
```

2. Overloading "evaluate" method:

```
Package udfpack;
import org.apache.hadoop.hive.ql.exec.UDF;
    public class AbsValue extends UDF {
        public Double evaluate(double value) {
            return Math.abs(value);
        }

    public long evaluate(long value){
        return Math.abs(value);
    }
```

```
hive > addjar /home/training/HiveUDFs/absval.jar
```

```
hive> SELECT AbsValue(20.65) from dummy
```

3. Say hello:

```
Package udfpack;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

class Sayhello extends UDF
 {
  public Text evaluate(Text Input)
   {
    return new Text("Hello" + input.toString());
  }
}
```

```
hive> add jar /home/training/HiveUDFS/sayhello.jar
hive> create temporary function sayhello as 'udfpack.Sayhello'
hive> SELECT sayhello('Hadoop') from dummy;
```

4. Convert to lower case:

```
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class lower extends UDF {
```

```
public Text evaluate(final Text s){
    if(s==null) { return null; }
    return new Text(s.toString().toLowerCase());
  }
}
```

```
hive> add jar /home/training/HiveUDFs/lower.jar
hive> SELECT lower('HADOOP') from dummy;
hive > SELECT lower(name) from employee;
```

UDF OUTPUT:

```
hive> seleet strip('hadoop','ha') from dummy;
OK
doop
Time taken: 0.131 seconds, fetched: 1 row(s)
hive> seleet strip
OK
hiveUDF
```