1. **Problem Scenario 1 : You have been given mysql DB with following details.**

**user=retail\_dba**

**password=cloudera**

**database=retail\_db**

**table=retail\_db.categories**

**jdbc URL = jdbc:mysql://quickstart:3306/retail\_db**

**Please accomplish following activities.**

1. **Connect my sql DB and check the content of the tables.**
2. **Copy “retail\_db.categories” table to hdfs, without specifying directory name.**
3. **Copy “retail\_db.categories” table to hdfs, in a directory name “categories\_target”.**
4. **Copy “retail\_db.categories” table to hdfs, in a warehouse directory name “categories\_warehouse”.**

**Solution 1 :**

**step l :connecting to existing mysql Database**

**mysql --user=retail\_dba --password=cloudera retail\_db**

**step 2: show all the available tables**

**show tables;**

**step 3 : View/count data from a table in mysql.**

**select count(1) from categories;**

**step 4 : check the currently available data in HDFS directory**

**hdfs dfs -ls**

**step 5: import single table (Without specifying directory).**

**sqoop import –connect \**

**jdbc:mysql://quickstart:3306/retail\_db \**

**--username=retail\_dba --password=cloudera --table=categories**

**Note : please check you don’t have space between before or after ‘=’ sign.**

**Sqoop uses the MapReduce framework to copy data from rdbms to hdfs**

**Step6 : read the data from one of the partition, created using above command.**

**Hdfs dfs -cat categories/part-m-00000**

**Step7 : specifying target directory in import command (we are using number of mappers =1, you can change accordingly)**

**Sqoop import –connect jdb:mysql://quickstart:3306/retail\_db \**

**--username=retail\_dba --password=cloudera --table=categories \**

**--target dir=categories\_target --m1**

**Step8 : check the content in one of the partition file.**

**Hdfs dfs -cat categories\_target/part-m-00000**

**Step 9 : specifying parent directory so that you can copy more than one table in a specified target directory. Command to specify warehouse directory.**

**Sqoop import --connect jdb:mysql://quickstart:3306/retail\_db \**

**--username=retail\_dba --password=cloudera --table=categories \**

**--warehouse-dir=categories\_waewhouse --m1**

**Step 10 : see the content in one of the file (partition)**

**Hdfs dfs -cat categories\_warehouse/categories/part-m-00000**

1. **Problem Scenario 2 : There is a parent organization called "Acmeshell Group Inc", which has two child companies named QuickTechie Inc and Hadoop exam inc.  
   Both compnaies employee information is given in two separate text tile as below. Please do the following activity for employee details.**

**quicktechie.txt**

**1.Alok,hyderabad**

**2,Krish,Hongkong   
3,Jyoti,Mumbai   
4,Atul,Banglore   
5,lshan,Gurgaon**

**hadoopexam.txt   
6,John,Newyork**

**7,alp2004,california  
8,tellme,Mumbai   
9,Gagan21 , Pune   
10,Mukesh,Chennai**

**1. Which command will you use to check all the available command line options on HDFS and How will you get the Help for individual command.   
2. Create a new Empty Directory named Employee using Command line. And also create an empty file named in it quicktechie.txt   
3. Load both companies Employee data in Employee directory (How to override existing file in HDFS).   
4. Merge both the Employees data in a Single file called MergedEmployee.txt, merged files should have new line character at the end of each file content.   
5. Upload merged-tile on HDFS and change the file permission on HDFS merged file, so that owner and group member can read and write, other user can read the file.   
6. Write a command to export the individual file as well as entire directory trom HDFS to local file System.**

**Solution : 2**

**Step1: Check all available command**

**hdfs dfs**

**step2: get help on individual command**

**hdfs dfs -mkdir employee**

**Step 3: create a directory in HDFS using named employee and create a dymmy file in it called e.g. quickgechie.txt**

**hadoop fs -touchz employee/quickgechie.txt**

**now create an empity file in employee directory using Hue.(as shown in video)**

**step 4: Create a directory on local file system and then create two files, with the given data in problems.**

**Step5: now we have an existing directory with content in it, now using HDFS command line, overrid this existing employee directory.**

**cd /home/cloudera/desktop/**

**hdfs dfs -put -f employee/quickgechie.txt**

**Step6: Check all files in directory copied successfully hdfs dfs -ls employee**

**Step7: now merge all the files in employee directory.**

**hdfs dfs -getmerge -nl employee mergedemployee.txt**

**Step8: Check the content of the file.**

**cat mergedemployee.txt**

**Step9: Copy merged file in employed directory from local file system to hdfs.**

**hdfs dfs -put mergedemployee.txt employee/**

**Step10: Check file coped or not.**

**hdfs dfs -ls employee**

**Step11: Change the permission of the merged file on hdfs**

**hdfs dfs -chmod 664 employee/mergedemployee.txt**

**Step12: get the file from hdfs to local file system.**

**hdfs dfs -get employee employee-hdfs**

**====================================================================**

1. **Problem Scenario 3: You have been given MySQL DB with following details.   
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   table=retail\_db.categories   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db**

**Please accomplish following activities.   
1. Import data from catagories table, where category=22 (Data should be stored in categories\_subset)   
2. Import data from catagories table, where catagory>22 (Data should be stored in categories\_subset\_2)   
3. Import data from catagories table, where catagory between 1 and 22 (Data should be stored in categories\_subset\_3)   
4. While importing catagories data change the delimiter to '|' (Data should be stored in categories\_subset\_6)   
5. Importing data from catagories table and restrict the import to category\_name,category\_id columns only with delimiter as '|'   
6. Add null values in the table using below SQL statement   
ALTER TABLE categories modify category\_department\_id int(11);   
INSERT INTO categories values(60,null, ‘testing’);   
7. Importing data from catagories table (In categories\_subset\_17 directory) using '|' delimiter and category\_id between 1 and 61 and encode null values for both string and non string columns.   
8. Import entire schema retail\_db in a directory categories\_subset all tables**

**Solution 3 :**

**Step 1: Import Single table (Subset data) Note:**

**Here the • is the same you find on ~ key   
sqoop import --connect jdb:mysql://quickstart:3306/retail\_db \**

**--username=retail\_dba --password=cloudera --table=categories \**

**--warehouse-dir=categories\_subset –-where “category\_id=22” –m 1**

**Step 2 : Check the output partition   
hdfs dfs -cat categories\_subset/categories/part-m-00000   
Step 3 : Change the selection criteria (Subset data)   
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db \**

**--username--retail\_dba --password=cloudera -table=categories \**

**--warehouse-dir=categories\_subset\_2 –where “category\_id>22” -m 1  
Step 4 : Check the output partition   
hdfs dfs -cat categories\_subset\_2/categories/part-m-00000   
Step 5 : use between clause (Subset data)   
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_dba \**

**--username--retail\_dba --password=cloudera -table=categories \**

**--warehouse-dir=categories\_subset\_3 –where “category\_id between 1 and 22” -m 1   
Step 6 : Check the output partition   
hdfs dfs -cat categories\_subset\_3/categories/part-m-00000   
Step 7 : Changing the delimiter during import.   
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db \**

**--username--retail\_dba --password=cloudera --table=categories \**

**--warehouse-dir=categories\_subset\_3 –where ”category\_id between 1 and 22” --fields-terminated-by=”|” -m 1   
Step 8 : Check-the output partition   
hdts dts -cat categories\_subset\_6/categories/part-m-00000   
Step 9 : Selecting subset columns   
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db \**

**–-username=retail\_dba --password=cloudera --table=categories \**

**--warehouse-dir=categories\_subset\_col \**

**--where “category\_id between 1 and 22” \**

**--fields-terminated-by=”|” columns=”category\_name,category\_id” -m 1   
Step 10 : Check the output partition   
hdts dts -cat categories subset col/categories/part-m-00000**

**Step 11: inserting record with null values (using mysql)**

**Alter table categories modify category --department\_id int(11);**

**Insert into categories values (60, null,’testing’);**

**Select \* from categories;**

**Step 12: encode non string null column**

**Squoop import --connect jdbc:mysql://quickstart:3306/retail\_db \**

**–username=retail\_ dba --password=cloudera --table=categories \**

**--warehouse-dir=categories\_subset\_17 \**

**--where “category\_id between 1 and 61” \**

**--fields-terminated-by=’|’ --null-string=”//N” \**

**–-null-non-string=”//N” -m 1**

**Step 13: view the content**

**Hdfs dfs -cat categories\_subset\_17/categories/part-m-00000**

**Step 14 : Import all the tables from a schema(this step will take little time)**

**Sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db \**

**–-username=retail\_ dba --password=cloudera \**

**--warehouse-dir=categories\_subset\_all\_tables -m 1**

**Step 15 : view the contents**

**Hdfs dfs -ls categories\_subset\_All\_tables**

**Step 16 : cleanup or back to originals.**

**Delete from categories where category\_department\_id in(59,60);**

**Alter table categories modify category\_department\_id int (11) not null;**

**Alter table categories modify category\_name varchar(45) not null;**

**Desc categories;**

1. **Problem Scenario 4: You have been given MySQL DB with following details.   
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   table=retail\_db.categories   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Please accomplish following activities.   
   Import Single table categories(Subset data) to hive managed table where category\_id between 1 and 22**

**Solution 4 :**

**Step1 : Import single table (subset data)**

**Squoop import --connect jdbc:mysql://quickstart:3306/retail\_db \**

**-–username=retail\_dba --password=cloudera --table=categories \**

**--where “category\_id between 1 and 22” \**

**--hive-import -m 1**

**Note: here the is the sand you find on ~ key**

**This command will create a managed table and content will be created in the following directory.**

**/user/hive/warehouse/categories**

**Step2 : check whether table is created or not (in Hive)**

**Show tables;**

**Select \* from categories;**

1. **Problem Scenario 5 : You have been given following mysql database details.   
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db**

**Please accomplish following activities.   
1. List all the tables using sqoop command from retail\_db   
2. Write simple sqoop eval command to check whether you have permission to read database tables or not.   
3. Import all the tables as avro files in /user/hive/warehouse/retail\_cca174.db   
4. Import departments table as a text tile in /user/cloudera/departments**

**Sloution 5:**

**Step 1 : list tables using sqoop**

**Sqoop list-tables -–conect jdbc:mysql://quickstart:3306/retail\_db \**

**--username retail\_ dba –-password cloudera**

**Step 2 : eval command, just run a count query on one of the table.**

**Sqoop eval \**

**--connect jdbc:mysql://quickstart:3306/retail\_db \**

**--username retail\_dba \**

**--password cloudera \**

**--query “select count(1) from order\_items”**

**Step 3: Sqoop import-all tables \**

**--connect jdbc:mysql://quickstart:3306/retail\_db \**

**--username retail\_dba \**

**--password cloudera \**

**--as-avrodatafile \**

**--warehouse-dir=/user/hive/warehouse/retail\_stage.db \**

**-m 1**

**Step 4: Import departments tables as a text in /user/cloudera/departments**

**Sqoop import \**

**--connectjdbc:mysql://quickstart:3306/retail\_db \**

**--username retail\_dba \**

**--password cloudera \**

**--table departments\**

**--as-textfile \**

**--target-dir=/user/clouera/departments**

**Step5 : verify the imported data.**

**Hdfs dfs -ls /user/cloudera/departments**

**Hdfs dfs -ls /user/hive/warehouse/retail\_stage.db**

**Hdfs dfs -ls /user/hive/warehouse/retail\_stage.db/products**

**====================================================================**

1. **Problem Scenario 6 : You have been given following mysql database details as well as other info.   
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Compression Codec : org.apache.hadoop.io.compress.snappycodec   
   Please accomplish following.   
   1. Import entire database such that it can be used as a hive tables, it must be created in default schema.   
   2. Also make sure each table files is partitioned in 3 files e.g. part-0000, part-00002, part-00003   
   3. Store all the java tiles in a directory called java\_output to evalute the further**

**6. Solution :**

**Step 1 : Drop all the tables, which we have created in previous problems. Betore implementing the solution.**

**Login to hive and execute following command.   
show tables;   
drop table categories;   
drop table customers;   
drop table** **departments;   
drop table employee;   
drop table order\_items;   
drop table orders;   
drop table products;   
show tables;   
Check warehouse directory.   
hdfs dfs -ls user/hive/warehouse**

**Step 2 : Now we have cleaned database. Import entire retail db with all the required parameters as problem statement is asking.   
sqoop import-all-tables \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail dba \   
--password=cloudera \   
--hive-import \   
--hive-overwrite \   
--compress**

**--compression-codec org.apache.hadoop.io.compress.snappycodec \   
--outdir java\_output**

**Step 3 : Verify the work is accomplished or not.   
a. Go to hive and check all the tables   
hive   
show tables;   
select count(l) from customers;   
b. Check the warehouse directory and number ot partitions.   
hdfs dfs -ls luser/hive/warehouse   
hdfs dfs -ls luser/hive/warehouse/categories   
c. Check the output java directory.   
ls -ltr java\_output/**

1. **Problem Scenario 7 : You have been given following mysql database details as well as other info.   
   User=retail\_dba   
   password=cloudera   
   database=retail db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Please accomplish following.   
   1.Import department tables using your custome boundary query, which import departments between 1 to 25.   
   2. Also make sure each tables file is partitioned in 2 tiles e.g. part-0000, part-00002   
   3. Also make sure you have imported only two columns trom table, which are department\_id,department\_name**

**Solutions : 7**

**Step 1 : Clean the hdfs file system, if they exists clean out.   
hadoop fs -rm -R departments   
hadoop fs -rm -R categories   
hadoop fs -rm -R products   
hadoop fs -rm -R orders   
hadoop fs -rm -R order\_itmes   
hadoop fs -rm -R customers   
Step 2 : Now import the department table as per requirement.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--target-dir /user/cloudera/departments \   
--boundary-query "select 1, 25 from departments" \   
--columns department\_id,department\_name -m 2**

**Step 3 : Check-imported data.   
hdfs dfs -Is departments   
hdfs dfs -cat departments/part-m-00000   
hdfs dfs -cat departments/part-m-00001**

**====================================================================**

1. **Problem Scenario 9 : You have been given following mysql database details as well as other into.   
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Please accomplish following.   
   1. Import departments table in a directory.   
   2. Again import departments table same directory (However, directory already exist hence it should not overrride and append the results)   
   3. Also make sure your results fields are terminated by '|' and lines terminated by '\n'.**

**9. Solutions :   
Step 1 : Clean the hdfs tile system, it they exists clean out.   
hadoop fs -rm -R departments   
hadoop fs -rm -R categories   
hadoop fs -rm -R products   
hadoop fs -rm -R orders   
hadoop fs -rm -R order itmes   
hadoop fs -rm -R customers**

**Step 2 : Now import the department table as per requirement.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--target-dir=departments \   
--fields-terminated-by '|' \   
--lines-terminated-by '\n' \**

**-m 1**

**Step 3 : Check imported data.   
hdfs dfs -ls departments   
hdfs dfs -cat departments/part-m-00000**

**Step 4 : Now again import data and needs to appended.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \**

**--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--target-dir departments \   
--append \   
--fields-terminated-by '|' \   
--lines-terminated-by '\n' \   
-m 1**

**Step 5 : Again Check the results   
hdfs dfs -ls departments   
hdfs dfs -cat departments/part-m-00001**

**====================================================================**

1. **Problem Scenario : You have been given following mysql database details as well as other into.**

**User=retail=dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish following.   
1. Create a database named hadoopexam and then create a table named departments in it, with following fields.   
department\_id int,   
department\_name string   
e.g. location should be hdfs://quickstart.cloudera:8020/user/hive/warehouse/hadoopexam.db/departments   
2. Please import data in existing table created above from retai\_db.departments into hive table hadoopexam.departments.   
3. Please import data in a non-existing table, means while importing create hive table named hadoopexam.departments\_new**

**Solution : 10**

**Step 1 : Go to hive interface and create database.   
hive   
create database hadoopexam;   
Step 2. use the database created in above step and then create table in it.   
use hadoopexam;   
show tables;   
Step 3 : Create table in it.   
create table departments (department\_id int, department\_name string);   
show tables;   
desc departments;   
desc Formatted departments;   
Step 4 : Please check following directory must not exist else it will give error.   
hdfs dfs -ls /user/cloudera/departments   
If directory already exists, make sure it is not useful and than delete the same.   
This is the staging directory where Sqoop store the intermediate data before pushing in hive table.   
hadoop fs -rm -R departments   
Step 5 : Now import data in existing table   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \**

**--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--hive-home /user/hive/warehouse \   
--hive-import \   
--hive-overwrite \   
--hive-table hadoopexam.departments   
Step 6 : Check whether data has been loaded or not.   
hive;   
use hadoopexam;   
show tables;   
select \* from departments;   
desc formatted departments;   
Stpe 7 : Import data in non-existing tables in hive and create table while importing.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail dba \   
--password=cloudera \   
--table departments \   
--hive-home /user/hive/warehouse \   
--hive-import   
--hive-overwrite \**

**--hive-table hadoopexam.departments\_new \   
--create-hive-table**

**Step 8 : Check whether data has been loaded or not.   
hive;   
use hadoopexam;   
show tables;**

**select \* trom departments \_ new;   
desc formatted departments\_new;**

**====================================================================**

1. **Problem Scenario 11 : You have been given following mysql database details as well as other info.  
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Please accomplish following.   
   1. Import departments table in a directory called departments.   
   2. Once import is done, please isert following 5 records in departments mysql table.   
   Insert into departments(10, physics);   
   Insert into departments(11, Chemistry);   
   Insert into departments(12, Maths);   
   Insert into departments(13, Science);   
   Insert into departments(14, Engineering);   
   3. Now import only new inserted records and append to existring directory , which has been created in first step.**

**Solution : 11  
Step 1 : Clean already imported data. (In real exam, please make sure you dont delete data generated from previous exercise).   
hadoop fs -rm -R departments**

**Step 2 : Import data in departments directory.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--target-dir /user/cloudera/departments**

**Step 3 : Insert the five records in departments table.   
mysql –user=retail\_dba --password=cloudera retail\_db   
Insert into departments values(10, "physics");   
Insert into departments values(11, "Chemistry");   
Insert into departments values(12, "Maths");   
Insert into departments values(13, "Science");   
Insert into departments values(14, "Engineering");   
commit;**

**select \* from departments;**

**Step 4 : Get the maximum value of departments from last import.   
hdfs dfs -cat /user/cloudera/departments/part\*   
that should be 7**

**Step 5 : Do the incremental import based on last import and append the results.   
sqoop import \   
--connect "jdbc:mysql://quickstart.cloudera:3306/retail\_db" \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--target-dir /user/cloudera/departments \   
--append \   
--check-column "department\_id" \   
--incremental append \   
--last-value 7**

**Step 6 : Now check the result.   
hdfs dfs -cat /user/cloudera/departments/part\***

1. **Problem Scenario 12 : You have been given following mysql database details as well as other info.   
   User=retail\_dba   
   password=cloudera   
   database=retail\_ db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Please accomplish following.   
   1. Create a table in retail\_db with following definition.   
   CREATE table departments\_new (department\_id int(11), department\_name varchar(45), created\_date TIMESTAMP DEFAULT NOW());  
   2. Now insert records from departments table to departments\_new   
   3. Now import data from departments \_ new table to hdfs.   
   4. Insert following 5 records in departments \_ new table.   
   Insert into departments \_ new values(110, "Civil" , null);   
   Insert into departments \_ new values(111, "Mechanical" , null);   
   Insert into departments \_ new values(112, "Automobile" , null);   
   Insert into departments \_ new values(113, "Pharma" , null);**

**Insert into departments \_ new values(114, "social engineering" , null);  
5. Now do the incremental import based on created\_date column.**

**Solution :**

**Step 1 : Login to musql db   
mysql --user=retail\_dba --password=cloudera   
show databases;   
use retail\_db;   
show tables;   
Step 2 : Create a table as given in problem statement.   
CREATE table departments\_new (department\_id int(11), department\_name varchar(45), created\_date TIMESTAMP DEFAULT NOW());   
show tables;   
Step 3 : isert records from departments table to departments \_ new   
insert into departments\_new select a.\*, null from departments a;   
Step 4 : Import data from departments\_new table to hdfs.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera**

**--table departments \_ new \**

**--target-dir /user/cloudera/departments\_new \   
--split-by department\_id   
Stpe 5 : Check the imported data.   
hdfs dfs -cat /user/cloudera/departments\_new/part\***

**Step 6: Insert following 5 records in departments \_ new table.   
Insert into departments \_ new values(110, "Civil" , null);   
Insert into departments \_ new values(111, "Mechanical" , null);   
Insert into departments \_ new values(112, "Automobile" , null);   
Insert into departments \_ new values(113, "Pharma" , null);   
Insert into departments\_new values(114, "Social Engineering" , null);   
commit;   
Stpe 7 : Import incremetal data based on created\_date column.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \_ new \   
--target-dir /user/cloudera/departments\_new \   
--append \   
--check-column created\_date \   
--incremental lastmodified \   
--split-by department\_id \   
--last-value "2016-01-30 12:07:37.0”**

**Step 8 : Check the imported value.   
hdfs dfs -cat /user/cloudera/departments\_new/part\***

1. **Problem Scenario 13 : You have been given following mysql database details as well as other info.   
   user=retail\_dba   
   password=cloudera   
   database=retail\_db   
   jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
   Please accomplish following.   
   1. Create a table in retail\_db with following definition.   
   CREATE table departments\_export (department\_id int(11), department\_name varchar(45), created\_date TIMESTAMP DEFAULT NOW());**

**2. Now import the data from following directory into departments\_export table.   
/user/cloudera/departments\_new**

**Solution : 13  
Step 1 : Login to musql db   
mysql –user=retail\_dba --password=cloudera   
show databases;   
use retail\_db;   
show tables;**

**Step 2 : Create a table as given in problem statement.**

**CREATE table departments\_export (department\_id int(11), department\_name varchar(45), created\_date TIMESTAMP DEFAULT NOW());   
show tables;**

**Step 3 : Export data from /user/cloudera/departments\_new to new table departments\_export   
sqoop export --connect jdbc:mysql://quickstart:3306/retail\_db \   
--username retail\_dba \   
--password cloudera \   
--table departments\_export \   
--export-dir /user/cloudera/departments\_new \   
--batch**

**Step 4 : Now check the export is correctly done or not.   
mysql —user=retail\_dba --password=cloudera   
show databases;   
use retail\_db;   
show tables;   
select \* from departments\_export;**

1. **Problem Scenario 14 : You have been given following mysql database details as well as other info.**

**User=retail\_dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish following activities.   
1. Create a csv file named updated\_departments.csv with the following contents in local file system.   
2,fitness   
3,footwear   
12,fathematics   
13,fcience   
14,engineering   
1000,management**

**2. Upload this csv file to hdfs filesystem,   
3. Now export this data from hdfs to mysql retail\_db.departments table. During upload make sure existing department will just updated and new departments needs to be inserted.  
4. Now update updated\_departments.csv file with below content.**

**2, Fitness   
3,Footwear   
12, Fathematics   
13,Science   
14,Engineering   
1000,Management   
2000,Quality Check**

**5.now upload this file to hdfs.**

**6.now export this data from hdfs to mysql retail\_db.departments table. During uploadmake sure existing department will just updated and no new departments needs to be inserted.**

**Solution 14 :**

**Step 1 : Create a csv file named updated\_departments.csv with give content.   
Step 2 : Now upload this file to HDFS.   
Create a directory called new\_data.   
hdfs dfs -mkdir new\_data   
hdfs dfs -put updated\_departments.csv new\_data/   
Step 3 : Check whether file is uploaded or not.   
hdfs dfs -Is new\_data   
Step 4 : Export this file to departments table using sqoop.   
sqoop export --connect jdbc:mysql://quickstart:330S/retail\_db \   
--username retail dba \   
--password cloudera   
--table departments \   
--export-dir new\_data \   
--batch \   
-m 1 \   
--update-key department\_id \   
--update-mode allowinsert   
Step 5 : Check whether required data upsert is done or not.   
mysql –user=retail\_dba —password=cloudera   
show databases;   
use retail\_db;   
show tables;   
select \* from departments;**

**Step 6 : update updated\_departments.csv file.   
Step 7 : Override the existing file in hdfs.   
hdfs dfs -put updated\_departments.csv new\_data/   
Step 8 : Now do the Sqoop export as per the requirement.   
sqoop export --connect jdbc:mysql://quickstart:3306/retail\_db \   
--username retail\_dba \   
--password cloudera \   
--table departments \   
--export-dir new\_data \   
--batch \   
-m 1 \   
--update-key department\_id \   
--update-mode updateonly   
Step 9 : Check whether required data update is done or not.   
mysql —user=retail\_dba --password=cloudera   
show databases;   
use retail\_db;   
show tables;   
select \* from departments;**

**15. Problem Scenario 15 : You have been given following mysql database details as well as other info.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish following activities.   
1. In mysql departments table please insert following record.   
Insert into departments values(9999, "'Data Science"' );   
2. Now there is a downstream system which will process dumps of this file. However, system is designed the way that it can process only files if fields are enclosed in(‘)single quote and separater of the field should be (~) and line needs to be terminated by : (colon).  
3. It data itself conatins the " (double quote ) than it should be escaped by \.  
4. Please import the departments table in a directory called departments\_enclosedby and file should be able to process by downstream system.**

**Solution : 15  
Step 1 : Connect to mysql database.   
mysql –user=retail\_dba--password=cloudera   
show databases;   
use retail\_db;   
show tables;   
Insert record   
Insert into departments values(9999, '"Data Science"' );   
select \* from departments;**

**Step 2 : Import data as per requirement.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--target-dir /user/cloudera/departments\_enclosedby \**

**—enclosed-by \' --escaped-by \\ --fields-terminated-by='~’ -lines-terminated-by:**

**Step 3 : Check the result.**

**hdfs dfs -cat /user/cloudera/departments\_enclosedby/part\***

**16. Problem Scenario 16 : You have been given following mysql database details as well as other info.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish below assignment.**

**1. Create a table in hive as below.   
create table departments\_hive(department\_id int, department\_name string);   
2. Now import data from mysql table departments to this hive table. Please make sure that data should be visible using below hive command.   
select • from departments\_hive**

**Solution :**

**Step 1 : Create hive table as said.   
hive   
show tables;   
create table departments\_hive(department\_id int, department\_name string);**

**Step 2 : The important here is, when we create a table without delimiter fields. Then default delimiter for hive is ^A (\001).**

**Hence, while importing data we have to provide proper delimiter.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--hive-home /user/hive/warehouse \   
--hive-import \   
--hive-overwrite \   
--hive-table departments\_hive \   
--fields-terminated-by '\001'**

**Step 3 : Check-the data in directory.   
hdfs dfs -Is /user/hive/warehouse/departments\_hive   
hdfs dfs -cat /user/hive/warehouse/departments\_hive/part\*   
Check data in hive table.   
select \* from departments\_hive;   
=================================**

**17. Problem Scenario 17 : You have been given following mysql database details as well as other info.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish below assignment.   
1. Create a table in hive as below.   
create table departments\_hive01(department\_id int, department\_name string, avg\_salary int);   
2. Create another table in mysql using below statement   
CREATE TABLE IF NOT EXISTS departments\_hive01(id int, department\_name varcnar(45), avg\_salary int);   
3. Copy all the data from departments table to departments\_hive01 using   
insert into departments\_hive01 select a.\*, null from departments a;   
Also insert following records as below   
insert into departments\_hive01 values(777, "Not known",1000);   
insert into departments\_hive01 values(8888, null,1000);   
insert into departments\_hive01 values(666, null,1100);   
4. Now import data from mysql table departments\_hive01 to this hive table. Please make sure that data should be visible using below hive command.  
Also, while importing if null value found for department\_name column replace it with “” (empty string) and for id column with -999   
select \* from departments\_hive;**

**Solution :**

**Step 1 : Create hive table as below.   
hive   
show tables;   
create table departments\_hive01(department\_id int, department\_name string, avg\_salary int);**

**Step 2 : Create table in mysql db as well.   
mysql --user=retail\_dba —password=cloudera   
use retail\_db   
CREATE TABLE IF NOT EXISTS departments\_hive01(id int, department\_name varchar(45),avg\_salary int);   
show tables;   
Step 3 : Insert data in mysql table.   
insert into departments\_hive01 select a.\*, null from departments a;   
check data inserts   
select \* from departments\_hive01 ;   
Now iserts null records as given in problem.   
insert into departments\_hive01 values(777, "Not known",1000);   
insert into departments\_hive01 values(8888, null,1000);   
insert into departments\_hive01 values(666, null,1100);   
Step 4 : Now import data in hive as per requirement.   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments\_hive01 \   
--hive-home /user/hive/warehouse \   
--hive-import \   
--hive-overwrite \   
--hive-table departments\_hive01 \   
--fields-terminated-by '\001' \   
--null-string “”\   
--null-non-string -999 \   
--split-by id \**

**-m 1**

**Step 5 : Check the data in directory.**

**hdfs dfs -Is /user/hive/warehouse/departments\_hive01   
hdfs dfs -cat/user/hive/warehouse/departments\_hive01/part\*   
Check data in hive table.   
select \* from departments\_hive01;**

**18. Problem Scenario 18 : You have been given following mysql database details as well as other info.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Now accomplish following activities.**

**1. Create mysql table as below.   
mysql --user =retail\_dba --password=cloudera   
CREATE TABLE IF NOT EXISTS departments\_hive02(id int, department\_name varcnar(45), avg\_salary int);   
show tables;   
2. Now export data from hive table departments\_hive01 in departments\_hive02. While exporting, please note following.   
wherever there is a empty string it should be loaded as a null value in mysql.   
wherever there is -999 value for int field, it should be created as null value.**

**Solution :**

**Step 1 : Create table in mysql db as well.   
mysql --user=retail\_dba —password=cloudera   
use retail\_db   
CREATE TABLE IF NOT EXISTS departments\_hive02(id int, department\_name varcnar(45), avg\_salary int);   
show tables;**

**Step 2 : Now export data from hive table to mysql table as per the requirement.   
sqoop export --connect jdbc:mysql://quickstart:3306/retail\_db \   
--username retail dba \   
--password cloudera \   
--table departments\_hive02 \   
--export-dir /user/hive/warehouse/departments\_hive01 \   
--input-fields-terminated-by '\001' \   
--input-lines-terminated-by '\n' \   
--num-mappers 1 \   
--batch \   
--input-null-string "" \   
--input-null-non-string -999**

**Step 3 : Now validate the data.   
select \* from departments\_hive02;**

**19. Problem Scenario 19 : You have been given following mysql database details as well as other info.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Now accomplish following activities.   
1. Import departments table from mysql to hdfs as textfile in departments \_ text directory.   
2. Import departments table from mysql to hdfs as sequncefile in departments \_ sequence directory.   
3. Import departments table from mysql to hdfs as avro file in departments\_avro directory.   
4. Import departments table from mysql to hdfs as parquet file in departments \_ parquet directory.**

**Solution :**

**Step 1 : Import departments table from mysql to hdfs as textfile   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--as-textfile \**

**--target-dir=departments\_text   
verity imported data   
hdfs dfs -cat departments\_text/part\***

**Step 2 : Import departments table from mysql to hdfs as sequncefile   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--as-sequencefile \   
--target-dir=departments\_sequence**

**verity imported data   
hdfs dfs -cat departments\_sequence/part\***

**Step 3 : Import departments table from mysql to hdfs as sequncefile   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--as-avrodatafile \   
--target-dir=department\_avro**

**verity imported data   
hdfs dfs -cat departments\_avro/part\***

**Step 4 : Import departments table from mysql to hdfs as sequncefile   
sqoop import \   
--connect jdbc:mysql://quickstart:3306/retail\_db \   
--username=retail\_dba \   
--password=cloudera \   
--table departments \   
--as-parquefile \   
--target-dir=departments\_parquet**

**verity imported data   
hdfs dfs -cat departments\_parquet/part\***

**20. Problem Scenario 20 : You have been given MySQL DB with following details.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
table=retail\_db.categories   
jdbc URL = “jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish following activities.   
1.Write a Sqoop Job which will import "retail\_db.categories" table to hdfs, in a directory name "categories\_target\_job".**

**Step 1 : Connecting to existing MySQL Database   
mysql –user=retail\_dba --password=cloudera retail\_db**

**step 2 . Show all the available tables   
show tables;**

**Step 3 : Below is the command to create Sqoop Job (Please note that — import space is mandatory)   
sqoop job --create sqoop\_job \   
-- import \   
--connect “jdbc:mysql://quickstart:3306/retail\_db”\  
--username=retail\_dba \   
--password=cloudera \   
--table categories \   
--target-dir categories\_target\_job \   
--fields-terminated-by 'l' \   
--lines-terminated-by '\n'**

**Step 4 : List all the Sqoop Jobs   
sqoop job --list**

**Step 5 : Show details of the Sqoop Job   
sqoop job --show sqoop\_job**

**Step 6 : Execute the sqoop job   
sqoop job --exec sqoop\_job**

**Step 7 : Check the output of import job   
hdfs dfs -Is categories target\_job   
hdfs dfs -cat categories target\_job/part\***

**Problem Scenario 21 : You have been given log generating service as below.   
start\_logs (It will generate continuous logs)   
tail \_ logs (You can check , what logs are being generated)   
stop\_logs (It will stop the log servoce)   
Path where logs are generated using above service : /opt/gen\_logs/logs/access.log   
Now write a flume configuration file named flume1 .conf, using that configuration file dumps logs in HDFS file system in a directory called flume1   
Flume channel should have following property as well. After every 100 message it should be committed, use non-durable/faster channel and it should be able to hold maximu 1000 events.**

**Solution:**

**Step 1 : Create flume configuration file, with below configuration for source, sink and channel.   
#Define source , sink , channel and agent.   
agent1 .sources = source1   
agent1.sinks = sink1   
agent1 .channels = channel1**

**# Describe/configure source1   
agent1 .sources.source1 .type = exec   
agent1.sources.source1 .command = tail -F /opt/gen\_logs/logs/access.log**

**## Describe sink1   
agent1 .sinks.sink1.channel = memory-channel   
agent1 .sinks.sink1.type = hdFs   
agent1 .sinks.sink1.hdfs.path = flume1   
agent1.sinks.sink1.hdfs.fileType = DataStream**

**# Now we need to define channel1 property.   
agent1.channels.channel1.type = memory   
agent1.channels.channel1 .capacity = 1000**

**agent1.channels.channel1.transactionCapacity= 100**

**# Bind the source and sink to the channel   
agent1.sources.source1 .channels = channel1  
agent1.sinks.sink1.channel=channel1**

**Step 2 : Run below command which will use this configuration file and append data in hdfs.   
Start log service using : start\_logs   
Start flume service :   
flume-ng agent –conf/ home/cloudera/flumeconf --conf-file /home/cloudera/flumeconf/flume1 .conf -Dflume.root.logger=DEBUG,lNFO,console –name agent1  
Wait for few mins and than stop log service.**

**Stop\_logs**

**Problem Scenario 22 : You have been given below comma separated employee information.   
name, salary, sex, age**

**alok, 100000, male,29**

**jatin, 10500, male,32  
yogesh,134000,male,39   
ragini,112000,female,35   
jyotsana,129000,female,39   
Valmiki,123000,male,29**

**use the netcat service on port 44444, and nc above data line by line. Please do the following activities.   
1. Create a flume conf file using fastest channel, which write data in hive warehouse directory, in a table called flume employee (Create hive table as well for given data).  
2. Write a hive query to read average salary of all employees.**

**Solution :**

**Step 1 : Create hive table tor flumeemployee.'   
CREATE TABLE flumeemployee   
(**

**name string,   
salary int,   
sex string,   
age int   
)**

**ROW FORMAT DELIMITED   
FIELDS TERMINATED BY ‘,’;**

**Step 2 : Create flume configuration file, with below configuration for source, sink and channel and save it in flume2.conf.   
#Define source , sink , channel and agent.   
agent1.sources = source1**

**Agent1.sinks = sink1   
agent1.channels = channel1   
# Describe/configure source1   
agent1.sources.source1.type = netcat   
agent1.sources.source1.bind = 127.0.0.1   
agent1.sources.source1.port = 44444**

**## Describe sink1  
agent1.sinks.sink1.channel = memory-channel   
agent1.sinks.sink1.type = hdfs   
agent1.sinks.sink1.hdfs.path = /user/hive/warehouse/flumeemployee   
hdfs-agent.sinks.hdfs-write.hdfs.writeFormat=Text   
agent1.sinks.sink1.hdfs.filetype = DataStream**

**# Now we need to define channel1 property.   
agent1 .channels.channel1 .type = memory   
agent1 .channels.channel1 .capacity = 1000   
agent1 .channels.channel1. transaction capacity = 100**

**# Bind the source and sink to the channel   
Agent1 .sources.source1 .channels = channel1   
agent1.sinks.sink1 .channel = channel1**

**Step 3 : Run below command which will use this configuration file and append data in hdfs.   
Start flume service :   
flume-ng agent --conf /home/cloudera/flumeconf --conf-file /home/cloudera/flumecont/flume2.conf —name agent1  
Step 4 : Open another terminal and use the netcat service.   
nc localhost 44444   
Step 5 : Enter data line by line.   
alok,1000000,male,29**

**Jatin, 105000,male,32**

**yogesh,134000,male,39   
ragini,112000,female,35   
jyotsana,129000,female,39**

**Valmiki,123000,male,29   
Step 6 : Open hue and check the data is available in hive table or not.   
Step 7 : Stop flume service by pressing ctrl+c   
Step 8 : Calculate average salary on hive table using below query. You can use either hive command line tool or hue.   
select avq(salarv) from flumeemplovee;**

**Problem Scenario 23 : You have been given log generating service as below.   
start\_logs (It will generate continuous logs)   
tail \_ logs (You can check , what logs are being generated)   
stop\_logs (It will stop the log servoce)   
Path where logs are generated using above service : /opt/gen\_logs/logs/access.log   
Now write a flume configuration file named flume3.conf, using that configuration file dumps logs in HdFS file system in a directory called flume3/%y/%m/%d/%H%M  
(Means every minute new directory should be created). Please us the interceptors to provide timestamp information, it message header does not have header info.  
And also note that you have to preserve existing timestamp, it message contains it.   
Flume channel should have following property as well. After every 100 message it should be committed, use non-durable/faster channel and it should be able to hold maximum 1000 events.**

**Solution :   
Step 1 : Create flume configuration file, with below configuration for source, sink and channel.   
#Define source , sink , channel and agent.   
agent1.sources = source1   
agent1.sinks = sink1   
agent1.channels = channel1  
# Describe/configure source1**

**Agent1.sources.source1 .type = exec   
agent1.sources.source1.command = tail -F /opt/gen\_logs/logs/access.log   
#Define interceptors   
agent1.sources.sourcel.interceptors=i1**

**Agent1.sources.source1. interceptors.i1.type=timestamp   
agent1.sources.source1. interceptors.i1.preserveExisting=true   
## Describe sink1  
agent1.sinks.sink1.channel = memory-channel   
agent1.sinks.sink1.type = hdfs   
agent1.sinks. sink1.hdts.path = flume3/% y/%m/%d/%H%M  
agent1 .sinks.sink1 .hdfs.fileType = DataStream   
# Now we need to define channel1 property.   
agent1.channels.channel1.type = memory   
agent1.channels.channel1.capacity = 1000   
agent1 .channels.channel1 .transactioncapacity = 100**

**# Bind the source and sink to the channel   
agent1.sources.source1.channels = channel1   
agent1.sinks.sink1 .channel = channel1**

**Step 2 : Run below command which will use this configuration file and append data in hdfs.   
Start log service using : start\_logs   
Start flume service :   
flume-ng agent ---conf/home/cloudera/flumeconf —conf-file /home/cloudera/flumeconf/flume3.conf -Dflume.root.logger=DEBUG,lNFO,console --name agent1  
Wait for few mins and than stop log service.   
stop\_logs**

**=============================================================================**

**Problem Scenario 24 : You have been given below comma separated employee information.   
name,salary,sex,age   
alok,1000000,male,29**

**Jatin, 105000,male,32**

**yogesh,134000,male,39   
ragini,112000,female,35**

**jyotsana,129000,female,39   
Valmiki,123000,male,29**

**use the netcat service on port 44444, and nc above data line by line. Please do the following activities.**

**1. Create a flume conf file using fastest channel, which write data in hive warehouse directory, in a table called flumemaleemployee (create hive table as well for given data).  
2. While importing, make sure only male employee data is stored.**

**Solution :   
Step 1 : Create hive table for flumeemployee.'   
CREATE TABLE numemaleemployee   
{**

**name string,   
salary int,   
sex string,   
age int   
}**

**ROW FORMAT DELIMITED   
FIELDS TERMINATED BY ‘,’;**

**Step 2 : Create flume configuration file, with below configuration for source, sink and channel and save it in flume4.conf.   
#Define source , sink , channel and agent.   
agent1.sources = source1   
agent1.sinks = sink1   
agent1.channels = channel1   
# Describe/configure source1   
Agent1.sources.source1 .type = netcat   
agent1.sources.source1.bind = 127.0.0.1   
agent1. sources. Source1. port = 44444   
#Define interceptors   
agent1.sources. source1.interceptors=i1   
agent1.sources.source1.interceptors.i1.type=regex\_filter   
agent1.sources.source1.interceptors.i1. regex=female   
agent1.sources.source1. interceptors.i1.excludeEvents=true**

**## Describe sink1  
agent1.sinks.sink1.channel = memory-channel   
agent1.sinks.sink1.type = hdfs   
agent1.sinks.sink1.hdfs.path = /user/hive/warehouse/flumemaleemployee   
hdfs-agent.sinks.hdfs-write.hdfs.writeFormat=Text   
agent1.sinks.sink1.hdfs.fileType = DataStream**

**# Now we need to define channel1 property.   
agent1.channels.channel1.type = memory   
agent1.channels.channel1.capacity = 1000   
agent1.channels.channel1.transactioncapacity = 100**

**# Bind the source and sink to the channel   
agent1.sources.source1.channels = channel1   
agent1.sinks.sink1.channel = channel1**

**Step 3 : Run below command which will use this configuration file and append data in hdfs.   
Start flume service :   
flume-ng agent --conf /home/cloudera/flumeconf --conf-file /home/cloudera/flumeconf/flume4.conf --name agent1**

**Step 4 : Open another terminal and use the netcat service.   
nc localhost 44444**

**Step 5 : Enter data-line by line.   
alok,1000000,male,29**

**Jatin, 105000,male,32**

**yogesh,134000,male,39   
ragini,112000,ftemale,35**

**jyotsana,129000,temale,39   
Valmiki,123000,male,29**

**Step 6: Open hue and check the data is available In hive table or not.**

**Step7: stop flume service by pressing ctrl+c**

**Step8 : Calculate average salary on hive table using below query. You can use either hive command line tool or hue.**

**Select avg(salary)from flumeemployee;**

**==============================================================================**

**Problem Scenario 25 : You have been given below comma separated employee information. That needs to be added in /home/cloudera/flumetest/in.txt file (to do tail source)   
sex,name,city   
1,alok,mumbai   
1,jatin,chennai   
1,yogesh,kolkata   
2,ragini,delhi   
2,jyotsana,pune   
1,valmiki,banglore**

**Create a flume conf file using fastest non-durable channel, which write data in hive warehouse directory, in two separate tables called flumemaleemployee1 and flumefemaleemployee1   
(Create hive table as well tor given data). Please use tail source with /home/cloudera/flumetest/in.txt file.   
flumemaleemployee1 : will contain only male employees data   
flumetemaleemployee1 : Will contain only woman employees data**

**=====================================================================**

**Solution :   
Step 1 : Create hive table tor flumemaleemployee1 and .'   
CREATE TABLE flumemaleemployee1  
{**

**Sex\_type int ,   
name string,   
city string   
}**

**ROW FORMAT DELIMITED   
FIELDS TERMINATED BY ‘,’ ;**

**CREATE TABLE flumefemaleemployee1  
{**

**Sex\_type int ,   
name string,   
city string   
}**

**ROW FORMAT DELIMITED   
FIELDS TERMINATED BY ‘,’;   
Step 2 : Create below directory and file   
mkdir /home/cloudera/flumetest/   
cd /home/cloudera/flumetest/   
Step 3 : Create flume configuration file, with below configuration for source, sink and channel and save it in flume5.conf.   
agent.channels = mem1 mem2   
agent.sinks = std1 std2   
agent.sources.tailsrc.type = exec   
agent.sources.tailsrc.command = tail -F /home/cloudera/flumetest/in.txt   
agent.sources.tailsrc.batchSize = 1**

**agent.sources.tailsrc.interceptors = i1   
agent.sources.tailsrc.interceptors.i1 .type = regex\_extractor   
agent.sources.tailsrc.interceptors.il .regex = ^(\\d)   
agent.sources.tailsrc.interceptors.il .serializers = t1  
agent.sources.tailsrc.interceptors.il. serializers.t1 .name = type   
agent. sources.tailsrc.selector.type = multiplexing**

**agent.sources.tailsrc.selector.header = type   
agent.sources.tailsrc.selector.mapping.l = mem1  
agent.sources.tailsrc.selector.mapping.2 = mem2   
agent.sinks.std1.type = hdfs   
agent.sinks.std1 .channel = mem1  
agent.sinks.std1.batchSize = 1   
agent.sinks.std1 .hdfs.path = /user/hive/warehouse/flumemaleemployee1  
agent.sinks.std1 .rolllnterval = 0   
agent.sinks.std1 .hdfs.fileType = DataStream**

**agent.sinks.std2.type = hdfs   
agent.sinks.std2.channel = mem2   
agent.sinks.std2.batchSize = 1   
agent.sinks.std2.hdts.path = /user/hive/warehouse/flumetemaleemployee1  
agent.sinks.std2.rolllnterval = 0   
agent.sinks.std2.hdfs.fileType = DataStream   
agent.channels.mem1.type = memory   
agent.channels.mem1.capacity = 100   
agent.channels.mem2.type = memory   
agent.channels.mem2.capacity = 100   
agent.sources.tailsrc.channels = mem1 mem2**

**Step 4 : Run below command which will use this configuration file and append data in hdfs.   
Start flume service :   
flume-ng agent --conf /home/cloudera/flumeconf --cont-file /home/cloudera/flumecont/flume5.conf --name agent   
Step 5 : Open another terminal create a file at /home/cloudera/flumetest/in.txt .   
Step 6 : Enter below data in tile and save it.   
1,alok,mumbai   
1,jatin,chennai   
1,yogesh,kolkata   
2,ragini,delhi   
2,jyotsana,pune   
1,valmiki,banglore**

**Step 7: open hue and check the data is available in hive table or not.**

**Step8: Stop flume service by pressing ctrl+c**

**==========================================================================================================================================================================**

**Problem Scenario 26 : You need to implement near real time solutions for collecting information when submitted in file with below information.   
You have been given below directory location (if not available than create it) /tmp/nrtcontent . Assume your departments upstream service is continuously committing data, because it is near real time solution). As soon as file committed in this directory that needs to be available in hdfs in /tmp/flume location.   
Write a flume configuration file named flume6.conf and use it to load data in hdfs with following additional properties .   
1. Spool /tmp/nrtcontent**

**2. File prefix in hdfs sholuld be events   
3. File suffix should be .log   
4. If file is not commited and in use than it should have\_as prefix.  
5. Data should be written as text to hdfs**

**===================================================================================**

**Solution :   
Step 1 : Create directory   
mkdir /tmp/nrtcontent   
step 2 : Create flume configuration file, with below configuration for source, sink and channel and save it in flume6.conf.   
agent1.sources = source1  
agent1.sinks = sink1   
agent1.channels = channel1**

**agent1.sources.source1 .channels = channel1   
agent1.sinks.sink1.channel = channel1**

**agent1 .sources.source1.type = spooldir   
agent1. sources. Source1.spoolDir = /tmp/nrtcontent**

**agent1 .sinks.sink1.type = hdfs   
agent1.sinks.sink1.hdfs.path = /tmp/flume   
agent1.sinks.sink1.hdfs.fileprefix = events   
agent1.sinks.sink1.hdfs.fileSuffix = .log   
agent1.sinks.sink1.hdfs.inuseprefix = \_  
agent1.sinks.sink1.hdfs.fileType = DataStream**

**agent1.channels.channel1.type = file**

**step4: Run below command which will use this configuration file and append data in hdfs.   
Start flume service :   
flume-ng agent --cont /home/cloudera/flumeconf --conf-file /home/cloudera/flumeconf/flume6.conf --name agent1**

**Step 5 : Open another terminal and create a file in /tmp/nrtcontent   
echo "l am preparing for CCA175 from HadoopExam.com" > /tmp/nrtcontent/.he1 .txt   
mv /tmp/nrtcontent/.he1 .txt /tmp/nrtcontent/he1 .txt**

**After few mins   
echo "l am preparing tor CCA175 from QuickTechie.com" > /tmp/nrtcontent/.qt1 .txt   
mv /tmp/nrtcontent/.qt1 .txt /tmp/nrtcontent/qt1.txt**

**=====================================================================================Problem Scenario 27 : You need to implement near real time solutions for collecting information when submitted in file with below information.   
You have been given below directory location (if not available than create it) /tmp/spooldir . You have a finacial subscription for getting stock prices from Bloomberg as well as  
Reuters and using ftp you download every hour new files from their respective ftp site in directories /tmp/spooldir/bb and /tmp/spooldir respectively.   
As soon as tile committed in this directory that needs to be available in hdfs in /tmp/flume/finance location in a single directory.   
Write a flume configuration file named flume7.conf and use it to load data in hdfs with following additional properties .   
1. Spool /tmp/spooldir/bb and /tmp/spooldir/dr   
2. File prefix in hdfs sholuld be events   
3. File suffix should be .log   
4. If file is not commited and in use than it should have\_as prefix  
5. Data should be written as text to hdfs.**

**=====================================================================**

**Solution :**

**Step 1 : Create directory   
mkdir /tmp/spooldir/bb**

**mkdir /tmp/spooldir/dr   
step 2 : Create flume configuration file, with below configuration for source, sink and channel and save it in flume7.conf.   
agent1 .sources = source1 source2   
agent1 .sinks = sink1   
agent1 .channels = channel1  
agent1 .sources.source1 .channels = channel1   
agent1 .sources.source2.channels = channel1  
agent1 .sinks.sink1 .channel = channel1   
agent1 .sources.source1 .type = spooldir   
agent1 .sources.source1 .spoolDir = /tmp/spooldir/bb   
agent1 .sources.source2.type = spooldir   
agent1. sources. source2.spoolDir = /tmp/spooldir/dr   
agent1 .sinks.sink1 .type = hdfs**

**agent1 .sinks.sink1 .hdfs.path = /tmp/flume/finance   
agent1 .sinks.sink1.hdfs.filePrefix = events   
agent1 .sinks.sink1 .hdfs.fileSuffix = .log   
agent1 .sinks.sink1 .hdfs.inuseprefix =\_   
agent1 .sinks.sink1 .hdfs.fileType = DataStream**

**agent1 .channels.channel1 .type = file**

**Step 4 : Run below command which will use this configuration file and append data in hdfs.   
Start flume service :   
flume-ng agent --conf /home/cloudera/flumeconf --conf-file /home/cloudera/flumeconf/flume7.conf –name agent1   
Step 5 : Open another terminal and create a file in /tmp/spooldir/   
echo “IBM ,100.20160104”>> “/tmp/spooldir/bb/.bb.txt   
echo”IBM,103.2016010” >> /tmp/spooldir/bb/.bb.txt   
mv /tmp/spooldir/bb/.bb.txt /tmp/spooldir/bb/bb.txt   
After few mins   
echo “IBM,100.2,20160104>> /tmp/spooldir/dr/.dr.txt   
echo “IBM,103.1,20160105”>> /tmp/spooldir/dr/.dr.txt   
mv /tmp/spooldir/dr/.dr.txt /tmp/spooldir/dr/dr.txt**

**=====================================================================================**

**Problem Scenario 28 : You need to implement near real time solutions for collecting information when submitted in file with below information.   
You have been given below directory location (it not available than create it) /tmp/spooldir2 .   
As soon as file committed in this directory that needs to be available in hdfs in /tmp/flume/primary as well as /tmp/flume/secondary location.   
However, note that /tmp/flume/secondary is optional, if transaction failed which writes in this directory need not to be rollback.   
Write a flume configuration file named flume8.conf and use it to load data in hdfs with following additional properties .**

**1. Spool /tmp/spooldir2 directory   
2. File prefix in hdfs sholuld be events   
3. File suffix should be .log   
4. If file is not commited and in use than it should have \_ as prefix.  
5. Data should be written as text to hdfs**

**===================================================================**

**Solution :**

**Step 1 : Create directory   
mkdir /tmp/spooldir2   
step 2 : Create flume configuration file, with below configuration for source, sink and channel and save it in flume8.conf.   
agent1 .sources = source1   
agent1.sinks = sink1a sink1b   
agent1 .channels = channel1a channel1b**

**agent1 .sources.source1 .channels = channel1a channel1b   
agent1 .sources.source1 .selector.type = replicating   
agent1. sources. Source1. selector. optional = channel1b   
agent1 .sinks.sink1a.channel = channel1a   
agent1 .sinks.sink1b.channel = channel1b   
agent1 .sources.source1 .type = spooldir   
agent1 .sources.source1 .spoolDir = /tmp/spooldir2**

**agent .sinks.sink1a.type = hdfs   
agent1 .sinks.sink1a.hdfs.path = /tmp/flume/primary   
agent1 .sinks.sink1a.hdfs.fileprefix = events   
agent1 .sinks.sink1a.hdfs.fileSuffix = .log   
agent1 .sinks.sink1a.hdfs.fileType = DataStream**

**agent1 .sinks.sink1a.type = hdfs   
agent1 .sinks.sink1a.hdfs.path = /tmp/flume/primary   
agent1 .sinks.sink1a.hdfs.filepretix = events   
agent1 .sinks.sink1a.hdfs.fileSuffix = .log   
agent1 .sinks.sink1a.hdfs.fileType = DataStream**

**agent1 .sinks.sink1b.type = hdfs   
agent1 .sinks.sink1b.hdfs.path = /tmp/flume/secondary   
agent1 .sinks.sink1b.hdfs.fileprefix = events   
agent1 .sinks.sink1b.hdfs.fileSuffix = .log   
agent1.sinks.sink1b.hdfs.fileType = DataStream**

**agent1 .channels.channel1a.type = file   
agent1 .channels.channel1b.type = memory**

**Step 4 : Run below command which will use this configuration file and append data in hdfs.   
Start flume service :**

**flume-ng agent --conf /home/cloudera/flumeconf --conf-file /home/cloudera/flumecont/flume8.conf --name agent1   
Step 5 : Open another terminal and create a file in /tmp/spooldir2/   
echo “IBM,100,20160104”>> /tmp/spooldir2/.bb.txt   
echo”IBM,103,20160105” >> /tmp/spooldir2/.bb.txt   
mv /tmp/spooldir2/.bb.txt /tmp/spooldir2/bb.txt**

**After few mins   
echo “IBM,100.2,20160104” >> /tmp/spooldir2/.dr.txt   
echo “IBM,103.1,20160105” >> /tmp/spooldir2/.dr.txt   
mv /tmp/spooldir2/.dr.txt /tmp/spooldir2/dr.txt**

**===========================================================================**

**Problem Scenario 29 : Please accomplish the following exercises using HDFS command line options.   
1. Create a directory in hdfs named hdfs commands.   
2. Create a tile in hdfs named data.txt in hdfs commands.   
3. Now copy this data.txt tile on local filesystem, however while copying tile please make sure file properties are not changed e.g. file permission   
4. Now create a tile in local directory named data\_local.txt and move this tile to hdfs in hdfs commands directory.   
5. Create a file data\_hdfs.txt in hdfs\_commands directory and copy it to local file system.   
6. Create a file in local filesystem named filel.txt and put it to hdfs**

**=====================================================================================**

**Solution :**

**Step 1 : Create directory**

**hdfs dfs -mkdir hdfs commands   
Step 2 : Create a file in hdfs named data.txt in hdfs commands.   
hdfs dfs -touchz hdfs\_commands/data.txt   
Step 3 : Now copy this data.txt file on local filesystem, however while copying file please make sure file properties are not changed e.g. file permissions.  
hdfs dfs -copyToLocal -p hdfs\_commands/data.txt /home/cloudera/Desktop/HadoopExam   
Step 4 : Now create a file in local directory named data\_local.txt and move this file to hdfs in hdfs\_commands directory.   
touch data\_local.txt   
hdfs dfs -moveFromLocal /home/cloudera/Desktop/HadoopExam/data\_local.txt hdfs\_commands/   
Step 5 : Create a file data\_hdfs.txt in hdfs commands directory and copy it to local file system.   
hdfs dfs -touchz hdfs commands/data\_hdfs.txt   
hdfs dfs -get hdfs\_-commands/data\_hdfs.txt /home/cloudera/Desktop/HadoopExam/   
Step 6 : Create a file in local filesystem named file1.txt and put it to hdfs   
touch file1.txt   
hdfs dfs -put /home/cloudera/Desktop/HadoopExam/file1 .txt hdfs\_commands/**

**===================================================================================**

**Problem Scenario 30 : You have been given three csv files in hdfs as below.   
EmployeeName.csv with the field (id, name)   
EmployeeManager.csv (id, managerName)   
Employeesalary.csv (id, Salary)   
using Spark and its API you have to generate a joined output as below and save as a text file (Separated by comma) tor final distribution and output must be sorted by id.  
Id,name,salary,managerName**

**=====================================================================================**

**Solution :**

**Step 1 : Create all three files in hdfs in directory called spark1(We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.  
Step 2 : Load EmployeeManager.csv file from hdfs and create PairRDDs   
val manager = sc.textFile("spark1/EmployeeManager.csv")   
val managerPairRDD = manager.map(x=> (x.split(" ,”)(0),x.split(“,”)(1)))**

**Step 3 : Load EmployeeName.csv file from hdfs and create PairRDDs   
val name = sc.textFile("spark1/EmployeeName.csv")   
val namePairRDD = name.map(x=> (x.split(",")(O),x.split(",”)(1)))**

**Step 4 : Load Employeesalary.csv file from hdfs and create PairRDDs   
val salary = sc.textFile("spark1/EmployeeSalary.csv")   
val salaryPairRDD = salary.map(x=> (x.split(",")(O),x.split(",”)(1)))**

**step 4 : Join all pairRDDS   
val joined = namePairRDD.join(salaryPairRDD).join(managerPairRDD)**

**Step 5 : Now sort the joined results.   
val joinedData = joined.sortByKey()**

**Step 6 : Now generate comma separated data.   
val finalData = joinedData.map(v=> (v.\_1, v.\_2.\_1.\_1,v.\_2.\_1.\_2,v.\_2.\_2))**

**Step 7 : Save this output in hdfs as text file.   
finalData.saveAsTextFile("spark1/result.txt")**

**================================================================**

**Problem Scenario 31 : You have given following two files   
1. Content.txt : Contain a huge text file containing space separated words.   
2. Remove.txt : Ignore/filter all the words given in this file (Comma Separated).   
Write a Spark program which reads the Content.txt tile and load as an ROD, remove all the words from a broadcast variables (which is loaded as an RDDof words from remove.txt). and count the occurrence of the each word and save ie as a text file HDFS.  
  
==================================================================**

**Solution :**

**Step 1 : Create all three files in hdfs in directory called spark2 (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.  
Step 2 : Load the Content.txt file   
val content = sc.textFile("spark2/Content.txt") //Load the text file   
Step 3 : Load the Remove.txt file   
val remove = sc.textFile("spark2/Remove.txt") //Load the text file   
Step 4 : Create- an RDD from remove, However, there is a possibility each word could have trailing spaces, remove these whitespaces as well.We have used two functions here flatMap,map and trim.  
val removeRDD= remove.flatMap(x=> x.split(",") ).map(word=>word.trim)//Create an array of words**

**Step 5 : Broadcast the variable, which you want to ignore   
val bRemove = sc.broadcast(removeRDD.collect().toList) // It should be array ot Strings**

**Step 6 : Split the content ROD, so we can have Array of String.   
val words = content.flatMap(line => line.split(“ ” ))**

**Step 7 : Filter the RDD, so it can have only content which are not present in "Broadcast Variable".   
val filtered = words.filter{case (word) => !bRemove.value.contains(word)}**

**Step 8 : Create a PairRDD, so we can have (word,1) tuple or PairRDD.   
val pairRDD = filtered.map(word => (word, 1))**

**Step 9 : Now do the word count on PairRDD.   
val wordCount = pairRDD.reduceByKey(\_ + \_)**

**Step 10 : Save the output as a Text file.   
wordCount.saveAsTextFile("spark2/result.txt")**

**===============================================================================**

**Problem Scenario 32 : You have given three files as below.   
spark3/sparkdir1/file1 .txt   
spark3/sparkdir2/file2.txt   
spark3/sparkdir3/file3.txt   
Each file contain some text. As given in RHS (Righ hand side).   
Now write a Spark code in scala which will load all these three files from hdfs and do the word count by filtering following words.   
And result should be sorted by word count in reverse order.   
Filter words (“a”,”the”,”an”, “as”, “a”, “with”, “this”, “this”, “these”, “is”, “are”, “in”, “for”, “to”, “and”, “the”, “of”   
Also please make sure you load all three files as a Single RDD (All three files must be loaded using single API call).   
You have also been given following codec   
import org.apache.hadoop.io.compress.GzipCodec   
Please use above codec to compress tile, while saving in hdfs.**

**Solution :   
Step 1 : Create all three files in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.**

**Step 2 : Load content from all files.   
valcontent=sc.textFile("spark3/sparkdir1/file1.txt,spark3/sparkdir2/file2.txt,spark3/sparkdir3/file3.txt") //Load the text file   
Step 3 : Now create split each line and create RDD of words.   
val flatContent = content.flatMap(word=>word.split(" "))**

**Step 4 : Remove space after each word (trim it)   
val trimmedContent = flatcontent.map(word=>word.trim)**

**Step 5 : Create an RDD from remove, all the words that needs to be removed.**

**val removeRDD =sc.parallelize(List(“a”, “the”, “an”, "as", “a”, “with”, “this”, “these”, “is”, “are”, “in”,"for", “to”, “and”, “the”, “of”))**

**Step 6 : Filter the RDD, so it can have only content which are not present in removeRDD.   
val filtered = trimmedContent.subtract(removeRDD)**

**Step 7 : Create a PairRDD, so we can have (word,1) tuple or PairRDD.   
val pairRDD = filtered.map(word => (word,1))**

**Step 8 : Now do the word count on PairRDD.   
val wordCount = pairRDD.reduceByKey(\_ + \_)**

**step 9 : NOW swap PairRDD.   
val swapped = wordCount.map(item => item. swap)**

**Stepn 10 : Now revers order the content.   
val sortedOutput = swapped. sortByKey(false)**

**Step 11 : Save the output as a Text file.   
sortedOutput.saveAsTextFile("spark3/result")**

**Step12:Save compressed output.**

**Import org.apache.hadoop.io.compress.Gzipcodec**

**sortedOutput.saveAsTextFile(“spark3/compressedresult”,classOf[GZipCodec])**

**===============================================================================**

**Problem Scenario 33 : You have given a files as below.**

**spark5/EmployeeName.csv (id,name)   
spark5/EmployeeSalary.csv (id,salary)   
Data is given in RHS (Righ hand side).   
Now write a Spark code in scala which will load these two files from hdfs and join the same, and produce the (name,salary) values.   
And save the data in multiple file group by salary (Means each tile will have name of employees with same salary). Make sure file name include salary as well.**

**==================================================================================**

**Solution :   
Step 1 : Create all three files in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.**

**Step 2 : Load EmployeeName.csv file from hdfs and create PairRDDs   
val name = sc.textFile("spark5/EmployeeName.csv")   
val namePairRDD = name.map(x=> (x.split(",")(O),x.split(",”)(1)))**

**Step 3 : Load Employeesalary.csv file from hdfs and create PairRDDs   
val salary = sc.textFile("spark5/EmployeeSalary.csv")   
val salaryPairRDD = salary.map(x=> (x.split(",")(O),x.split(",”)(1)))   
step 4 : Join all pairRDDS   
val joined = namePairRDD.join(salaryPairRDD)   
Step 5 : Remove key from RDD and Salary as a Key.   
val keyRemoved = joined.values   
Step 6 : Now swap filtered RDD.   
val swapped = keyRemoved.map(item => item. swap)   
Stepn 7 : Now groupBy keys (It will generate key and value array)   
val grpByKey = swapped.groupByKey().collect()   
Step 8 : Now create RDD tor values collection   
val rddByKey = grpByKey.map{case (k,v) => k->sc.makeRDD(v.toSeq)}   
Step 9 : Save the output as a Text file.   
rddByKey.foreach{ case (k,rdd) => rdd.saveAsTextFile("spark5/Employee"+k)}**

**==============================================================================**

**Problem Scenario 34 : You have given a tile named spark6/user.csv.   
Data is given in RHS (Righ hand side).   
Now write a Spark code in scala which will remove the header part and create ROD of values as below, for all rows. And also if id is "myself" than filter out row.**

**Map(id 0m, topic scala, hits 120)**

**Info : Dell-pc P security Code :1100148121513   
============================================================================**

**Solution :**

**Step 1 : Create file in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.   
Step 2 : Load user.csv file from hdfs and create PairRDDs   
val csv = sc.textFile("spark6/user.csv")   
Step 3 : split and clean data   
val headerAndRows = csv.map(line => line.split(",").map(\_.trim))   
Step 4 : Get header row   
val header = headerAndRows.first   
Step 5 : Filter out header (We need to check it the first val matches the first header name)   
val data = headerAndRows.filter(\_(O)!= header(O))   
Step 6 : Splits to map (header/value pairs)**

**val maps = data.map(splits => header.zip(splits).toMap)**

**Stepn 7 : Filter out the user "myself"   
val result = maps.filter(map => map("id") "myself")   
Step 8 : Save the output as a Text file.   
result.saveAsTextFile("spark6/result.txt")**

**=====================================================================================**

**Problem Scenario 35 : You have been given a file named spark7/EmployeeName.csv (id,name).   
1. Load this file from hdfs and sort it by name and save it back as (id,name) in results directory. However, make sure while saving it should be able to write in a single file.**

**==============================================================================**

**Solution :   
 Step 1 : Create file in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.   
Step 2 : Load EmployeeName.csv file from hdfs and create PairRDDs   
val name = sc.textFile("spark7/EmployeeName.csv")   
val namePairRDD = name.map(x=> (x.split(",")(O),x.split(",”)(1)))   
step 3 : NOW swap namepairRDD RDD.   
val swapped = namePairRDD.map(item => item.swap)   
Step 4 : Now sort the rdd by key.   
val sortedOutput = swapped.sortByKey()   
Step 5 : Now swap the result back   
val swappedBack =sortedOutput.map(item => item. swap)   
Step 6 : Save the output as a Text file and output must be written in a single file.   
swapped Back. repartition (1). saveAsTextFile(" spark7/result.txt")**

**===========================================================================**

**Problem Scenario 36 : You have been given a tile named spark8/data.csv (type,name).**

**1. Load this file from hdfs and save it back as (id, (all names of same type)) in results directory. However, make sure while saving it should be able to write in a single file.**

**============================================================================**

**Solution :   
Step 1 : Create file in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.   
Step 2 : Load data.csv file from hdfs and create PairRDDs   
val name = sc.textFile("spark8/data.csv")   
val namePairRDD = name.map(x=> (x.split(",")(O),x.split(",”)(1)))   
step 3 : NOW swap namepairRDD RDD.   
val swapped = namePairRDD.map(item => item.swap)**

**Step 4 : Now combine the rdd by key.**

**val combinedOutput = namePairRDD.combineByKey(List(\_), (x:List[String], y:String)=> y :: x, (x:List[String], y:List[String]) => x ::y)**

**Step 5 : Save the output as a Text file and output must be written in a single file.**

**combinedOutput. repartition (1). saveAsTextFile(" spark8/result.txt")**

**=====================================================================================**

**Problem Scenario 37 : HadoopExam.com has done survey on their Exam Products feedback using a web based form. With the following tree text field as input in web ui.  
Name : String   
Subscription Date : String   
Rating : String   
And servey data has been saved in a tile called spark9/feedback.txt   
Christopher|Jan 11, 2015|5**

**Kapil|11 jan, 2015|5  
Thomas|6/17/2014|5   
J0hn|22-08-2013|5   
Mithun|2013|5   
Jitendra||5   
Write a spark program using regular expression which will filter all the valid dates and save in two separate tile (good record and bad record)   
=====================================================================================**

**Solution :   
Step 1 : Create a file first using Hue in hdfs.   
Step 2 : Write all valid regular expressions sysntex tor checking whether records are having valid dates or not.**

**val reg1 = “””(\d+)\s(\w{3})(,)\s(\d{4})”””.r//11 Jan,2015**

**val reg2 = “””(\d+)(\/)(/d+)(\/)(/d{4})”””.r//6/17/2014**

**val reg3 = “””(\d+)(-)(\d+)(-)(\d{4})”””.r//22-08-2013**

**val reg4 = “””(\w{3})\s(\d+)(,)\s(\d{4})”””.r//Jan 11,2015  
Step 3 : Load the file as an RDD.   
val feedbackRDD = sc.textFile("spark9/feedback.txt")   
Step 4 : As data are pipe separated , hence split the same.   
val feedbackSplit = feedbackRDD.map(line => line.split('l'))   
Step 5 : Now get the valid records as well as , bad records.   
val validRecords = feedbackSplit.filter(x =>! (reg1.pattern.matches(x(1).trim).matches|reg2.pattern.matcher(x(1).trim).matches|reg3.pattern.matcher(x(1).trim).matchees|reg4.pattern.matcher(x(1).trim).matches))  
val badRecords = feedbackSplit.filter(x => ! (reg1.pattern.matches(x(1).trim).matches|reg2.pattern.matcher(x(1).trim).matches|reg3.pattern.matcher(x(1).trim).matchees|reg4.pattern.matcher(x(1).trim).matches))**

**Step 6 : Now convert each Array to Strings   
val valid =validRecords.map(e => (e(O),e(1),e(2)))   
val bad =badRecords.map(e => (e(O),e(1),e(2)))**

**Step 7 : Save the output as a Text file and output must be written in a single file.   
valid.repartition(1).saveAsTextFile("spark9/good.txt")   
bad. repartition saveAsTextFile(" spark9/bad.txt")**

**=======================================================================**

**Problem Scenario 38 : You have been given an RDD as below.   
val rdd: RDD[array[Byte]]  
Now you have to save this RDD as a SequenceFile. And below is the code snippet.   
import org.apache.hadoop.io.compress.GzipCodec   
rdd.map(bytesArray => (A.get(), new B(bytesArray))).saveAsSequenceFile ("/output/path",classOt[GzipCodec])   
What would be the correct replacement tor A and B in above snippet.**

**=======================================================================**

**Solution :   
A. NullWritable   
B. BytesWritable**

**========================================================================**

**Problem Scenario 39 : You have been given two files   
spark16/file1 .txt   
spark16/file2.txt   
1,9,5**

**2,7,4**

**3,8,3**

**Spark16/file2.txt**

**1,g,h**

**2,I,j**

**3,k,l  
Load these two files as Spark RDD and join them to produce the below results   
(1, ( (9,5), (g,h) ))   
(2, ( (7,4), (i,j) ))   
(3, ( (8,3), (k,l) ))   
And write code snippet which will sum the second columns of above joined results (5+4+3).   
  
==================================================================================**

**Solution :   
Step 1 : Create files in hdfs using Hue.**

**Step 2 : Create pairRDD tor both the files.   
val one =sc.textfile(“spark16/file2 .txt").map{   
\_.split(",", -1) match {   
case Array(a, b, c) => (a, ( b, c))**

**}**

**}**

**val two = sc.textFile("spark16/tile2.txt").map{**

**\_.split(",", -1) match {**

**case Array(a, b, c) => (a, (b, c))**

**}**

**}**

**step 3 : JOin both the RDD.   
val joined = one.join(two)**

**Step 4 : Sum second column values.   
val sum = joined.map {   
case (\_, ((\_, num2), (\_, \_))) => num2.tolnt   
}.reduce(\_ + \_)**

**===================================================================================**

**Problem Scenario 40 : You have been given sample data as below in a file called spark15/tile1 .txt   
3070811,1963,1096,, “US”, “CA”,,1,**

**3070811,1963,1096,, “US”, “CA”,,1,56**

**3070811,1963,1096,, “US”, “CA”,,1,23**

**Below is the code snippet to process this file.   
val field=sc.textFile(“spark15/file.txt”)   
val mapper = field.map(x=> A)   
mapper.map(x => x.map(x=> {B})).collect  
Please fill in A and B so it can generate below final output   
Array(Array(3070811, 1963,1096, 0, “US”, “CA”,,1, 0)**

**,Array(3022811, 1963,1096, 0, “US”, “CA”,,1, 56)  
,Array(3033811, 1963,1096, 0, “US”, “CA”,,1, 23)**

**Solution :**

**A. x.split(",", -1)   
B. if(x.isEmpty) 0 else x**

**=====================================================================================**

**Problem Scenario 41 : You have been given below code snippet.   
val au1 = sc.parallelize(List ( ("a" , Array(1 ,2)) , ("b" , Array(1 ,2))))   
val au2 = sc.parallelize(List ( ("a" , Array(3)) , ("b" , Array(2))))   
Apply the Spark method, which will generate below output.   
Array[(String, Array[lnt])] = Array((a,Array(1, 2)), (b,Array(1, 2)), (a,Array(3)), (b,Array(2)))**

**========================================================================**

**Solution   
Au1.union(au2)**

**=========================================================================**

**Problem Scenario 42 : You have been given a file (spark10/sales.txt), with the content as given in RHS.**

**And want to produce the output as a csv with group by Department,Designation,State with additional columns with sum(costToCompany) and TotalEmployeeCount**

**Should get result like**

**Dept,Desg,state,empCount,totalCost**

**Sales,Lead,AP,2,64000**

**Sales,Lead,LA,3,96000**

**Sales,Lead,TN,2,64000  
==================================================================================**

**Solution :**

**Step 1 : Create a file first using Hue in hdfs.   
step 2 : Load file as an RDD**

**val rawlines = sc.textFile("spark10/sales.txt")   
Step 3 : Create a case class, which can represent its column fileds.   
case class Employee(dep: String, des: String, cost: Double, state: String)**

**Step 4 : Split the data and create RDD of all Employee objects.   
val employees = rawlines.map(\_.split(",")).map(row =>Employee(row(0), row(1), row(2).toDouble, row(3)))   
Step 5 : Create a row as we needed. All group by fields as a key and value as a count for each employee as well as its cost.   
val keyVals = employees.map( em=> ((em.dep, em.des, em.state), (1 , em.cost)))   
Step 6 : Group by all the records using reduceByKey method as we want summation as well. For number of employees and their total cost.   
val results = keyVals.reduceByKey{ (a,b) => (a.\_1+ b.\_1, a.\_2 + b. \_2)} // (a.count + b.count , a.cost + b.cost )}   
Step 7 : Save the results in a text file as below.   
results. repartition (1). saveAsTextFile("spark10/group.txt")**

**===========================================================================**

**Problem Scenario 43 : You have been given following code snippet.   
val grouped = sc.parallelize(Seq(((1,"two"), List((3,4), (5,6)))))   
val flattened = grouped.flatMap { A=>   
groupValues.map { value => B }   
You need to generate following output. Hence replace A and B**

**Array((1,two,3,4),(1,two,5,6))  
===========================================================================**

**Solution :   
A case (key, groupValues)   
B (key.\_1, key.\_2, value.\_1, value.\_2)   
===========================================================================**

**Problem Scenario 44 : You have been given 4 files , with the content as given in RHS.   
(spark11/file1 .txt)   
(spark11/file2.txt)   
(spark11/file3.txt)   
(spark11/file4.txt)   
Write a Spark program, which will give you the highest occuring words in each file. With their file name and highest occuring words.**

**=============================================================================**

**Solution :**

**Step 1 : Create all 4 file first using Hue in hdfs.   
step 2 : Load all file as an RDD   
val file1 = sc.textFile("spark11/file1.txt")   
val file2 = sc.textFile("spark11/file2.txt")  
val file3 = sc.textFile("spark11/file3.txt")  
val file4 = sc.textFile("spark11/file4.txt")**

**Step 3 : Now do the word count for each file and sort in rverse order of count.   
val content1 = file1 .flatMap( line => line.split(" ")).map(word => (word,1)).reduceByKey(\_ + \_).map(item => item.swap).sortByKey(false).map(e=>e.swap)  
val content2 = file2.flatMap( line => line.split(" ")).map(word => (word,1)).reduceByKey(\_ + \_).map(item => item.swap).sortByKey(false).map(e=>e.swap)   
val content3 = file3.flatMap( line => line.split(" ")).map(word => (word,1)).reduceByKey(\_ + \_).map(item => item.swap).sortByKey(false).map(e=>e.swap)   
val content4 = file4.flatMap( line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_).map(item => item.swap).sortByKey(talse).map(e=>e.swap)**

**Step 4 : Split the data and create RDD ot all Employee objects.**

**val file1word = sc.makeRDD(Array(file1.name+"->"+content1(0).\_1+"-" +content1(0).\_2))  
val file2word = sc.makeRDD(Array(file2.name+"->"+content2(0).\_1+"-" +content2(0).\_2))   
val file3word = sc.makeRDD(Array(file3.name+"->"+content3(0).\_1+"-" +content3(0).\_2))   
val file4word = sc.makeRDD(Array(file4.name+"->"+content4(0).\_1+"-" +content4(0).\_2))   
step 5 : union all the RDDS   
val unionRDDs = file1word.union(file2word).union(file3word).union(file4word)   
Step 6 : Save the results in a text file as below.**

**unionRDDs.repartition(1).saveAsTextFile(“spark11/union.txt”)**

**====================================================================================**

**Problem Scenario 45 : You have been given 2 files , with the content as given in RHS.   
(spark12/technology.txt)   
(spark12/salary.txt)   
Write a Spark program, which will join the data based on first and last name and save the joined results in folloqing format.   
first,last,technology,salary**

**=====================================================================================**

**Solution :**

**Step 1 : Create 2 files first using Hue in hdfs.   
step 2 : Load all file as an RDD   
val technology = sc.textFile("spark12/technology.txt").map(e => e.split(","))   
val salary = sc.textFile("spark12/salary.txt").map(e => e.split(","))   
Step 3 : Now create Key,value pair ot data and join them.   
val joined = technology.map(e=>((e(0),e(1)),e(2))).jion(salary.map(e=>((e(0),e(1)),e(2))))  
Step 4 : Save the results in a text file as below.   
joined. repartition (1). saveAsTextFile("spark12/multiColumnJoined.txt")**

**====================================================================================**

**Problem Scenario 46 : You have been given belwo list in scala (name,sex,cost) for each work done.   
List( ("Deeapak" , "male", 4000), ("Deepak" , "male", 2000), ("Deepika" , "female", 2000),("Deepak" , "female", 2000), ("Deepak" , "male", 1000)   
Now write a Spark program to load this list as an RDD and do the sum of cost for combination of name and sex (as key)**

**===================================================================================**

**Solution :**

**Step 1 : Create an RDD out of this list   
val rdd = sc.parallelize(List( ("Deeapak" , "male”,4000), ("Deepaika" , "male" ,2000), ("Deepaika" , "female" ,2000), ("Deepak" , "female" ,2000), ("Deepak" , "male" ,1000), ("Neeta" , "female" ,2000)))  
step 2 : convert this RDD in pair RDD   
val byKey = rdd.map({case (name,sex,cost) => (name,sex)->cost})   
Step 3 : Now group by Key   
val byKeyGrouped = byKey.groupByKey   
Step 4 : Now sum the cost tor each group   
val result = byKeyGrouped.map{case ((id1,id2),values) => (id1,id2,values,sum)}  
Step 5 : Save the results   
result. repartition (1).saveAsTextFile(“spark12/result.txt”)**

**==================================================================================**

**Problem Scenario 47 : You have been given below code snippet, with intermediate output.**

**val z = sc.parallelize(List(1 ,2,3,4,5,6), 2)**

**// lets first print out the contents ot the RDD with partition labels**

**def myfunc(index: Int, iter:Iterator[(int)]) : Iterator[String] = {   
iter.toList.map(x => "[partlD:" + index + Val: " + x + "]").iterator   
//ln each run , output could be different, while solving problem assume belowm output only.   
z.mapPartitionsWithlndex(myfunc).collect   
res28: Array[String] =Array([partID:0, Val: 1], [partlD:O, Val: 2], [partlD:O, Val: 3], [partlD:1, Val: 4],   
[partlD:1, val: 5], [partlD:1, val: 6])   
Now apply aggreate method on ROD z , with two reduce function , first will select max value in each partition and second will add all the maximum values from all parttions  
Initialize the aggregate with value 5, hence expected output will be 16.**

**=====================================================================================**

**Solution :**

**z.aggregate(5)(math.max(\_, \_), \_+\_)**

**=====================================================================================**

**Problem Scenario 48 : You have been given below Python code snippet, with intermediate output.   
We want to take a list ot records about people and then we want to sum up their ages and count them.   
So tor this example the type in the RDD will be a Dictionary in the format of {name: NAME, age:AGE, gender:GENDER}.   
The result type will be a tuple that looks like so (Sum ot Ages, Count)   
people = O   
people.append({'name':'Amit', 'age':45,'gender':'M'})   
people.append({'name':'Ganga', 'age':43,'gender':'F'})   
people.append({'name':'John', 'age':28,'gender':'M'})   
people.append({'name':'Lolita', 'age':33,'gender':'F'})   
people.append({'name':'Dont Know', 'age':18,'gender':'T})   
peopleRdd=sc.parallelize(people) //Create an RDD   
peopleRdd.aggregate((0,0), seqOp, combOp)   
//Output ot above line : 167, 5)   
Now define two operation seqOp and combOp , such that   
seqOp : Sum the age ot all people as well count them, in each partition.   
combOp : Combine results from all partitions.**

**==================================================================================**

**Solution :**

**seqOp = (lambda x,y: (x[0] +y[‘age’],x[1] + 1))**

**combOp = (lambda x,y: (x[O] + Y[O], x[l] + Y[l]))**

**==================================================================================**

**Problem Scenario 49 : You have been given below code snippet (do a sum of values by key), with intermediate output.   
val keysWithValuesList = Array("foo=A", "foo=A", "foo=A", "foo=A", "foo=B", "bar=C", "bar=D", "bar=D")   
val data = sc.parallelize(keysWithValuesList)   
//Create key value pairs   
val kv = data.map(\_.split("=")).map(v => (v(0), v(1))).cache()   
val initialCount = 0;   
val countByKey = kv.aggregateByKey(initialCount)(addToCounts, sumPartitionCounts)   
Now define two functions (addToCounts, sumPartitionCounts) such, which will produce following results.   
Output 1   
countByKey.collect   
res3: Array[(String, Int)] = Array((foo,5), (bar,3))   
import scala.collection. \_  
val initialSet = scala.collection.mutable.HashSet.empty[String]   
val uniqueByKey = kv.aggregateByKey(initialSet)(addToSet, mergePartitionSets)**

**Now define two functions (addToSet, mergePartitionSets) such, which will produce following results.   
Output 2 :   
uniqueByKey.collect   
res4: Array[(String, scala.collection.mutable.HashSet[String])] = Array(( foo,set(B,A)),(bar,Set(C,D)))**

**==================================================================================**

**Solution :   
val addToCounts = (n: Int, v: String)=> n + 1   
val sumPartitionCounts = (P1: Int, p2: Int) => p1 + p2   
val addToSet = (s: mutable.HashSet[String], v: String) => s+= v   
val mergePartitionSets = (P1: mutable.HashSet[String], p2: mutable.HashSet[String]) => p1 ++= p2**

**==================================================================================**

**Problem Scenario 50 : You have been given below code snippet (calculating an average score), with intermediate output.   
type Scorecollector= (Int, Double)   
type Personscores = (String, (Int, Double))   
val initialscores = Array((“Fred”, 88.0), ("Fred", 95.0), ("Fred", 91.0), ("Wilma", 93.0), ("Wilma", 95.0), ("Wilma", 98.0))   
val wilmaAndFredScores = sc.parallelize(initialScores).cache()  
val scores = wilmaAndFredScores.combineByKey(createScoreCombiner, scorecombiner, scoreMerger)   
val averagingFunction = (personscore: Personscores) => {   
val (name, (numberscores, totalScore)) = personScore   
(name, totalScore / numberscores)   
val averagescores = scores.collectAsMap().map(averagingFunction)   
Expected output : averagescores: scala.conection.Map[string,Double] = Map(Fred-> 91.33333333333333, Wilma-> 95.33333333333333)   
Define all three required function , which are input tor combineByKey method. e.g. (createScoreCombiner, scorecombiner, scoreMerger).And help us producing required results**

**=====================================================================================**

**Solution :**

**val createScoreCombiner = (score: Double) => (1, score)   
val scorecombiner = (collector: Scorecollector, score: Double) => {   
val (numberscores, totalScore) = collector   
(numberscores + 1, totalScore + score)**

**}  
val scoreMerger= (collector1: Scorecollector, collector2: Scorecollector)=> {   
val (numScores1, totalScore1) = collector1  
val (numScores2, totalScore2) = collector2   
(numScores1 + numScores2, totalScore1 + totalScore2)   
}**

**Description :**

**The createScoreCombiner takes a double value and returns a tuple of (Int, Double)   
The scoreCombiner function takes a Scorecollector which is a type alias for a tuple of (Int,Double). We alias the values ot the tuple to numberScores and totalScore  
(sacraficing a one-liner for readablility). We increment the number of scores by one and add the current score to the total scores received so far  
The scoreMerger function takes two Scorecollectors adds the total number of scores and the total scores together returned in a new tuple.   
We then call the combineByKey tunction passing our previously defined functions.   
We take the resulting RDD, scores, and call the collectAsMap function to get our results in the form of(name,(numberScores,totalScore)).   
To get our final result we call the map function on the scores RDD passing in the averagingFunction which simply calculates the average score and returns a tuple-of(name,averageScore)**

**Calculating an average is a litte trickier compared to doing a count for the simple fact that counting is associative and commutative, we just sum all values for each partition and  
sum the partition values. But with averages, it's not that simple, an average of averages is not the same as taking an average across all numbes.But we can collect the total**

**number scores and total score per partition then divide the total overall score by the number of scores.**

**================================================================================**

**Problem Scenario 51 : You have been given below code snippet.   
val a = sc.parallelize(List(1, 2, 1, 3), 1)   
val b = a.map((\_, "b"))   
val c = a.map((\_, "c"))   
Operation\_xyz   
Write a correct code snippet for Operation\_xyz which will produce below output.   
Output :   
Array[(lnt, (Iterable[String], Iterable[String]))] = Array(   
(2,(ArrayBuffer(b),ArrayBuffer(c))),   
(3,(ArrayBuffer(b),ArrayBuffer(c))),   
(1,(ArrayBuffer(b, b),ArrayBuffer(c, c)))**

**================================================================================**

**Solution:   
b.cogroup(c).collect   
cogroup [Pair], groupWith [Pair]   
A very powerful set of functions that allow grouping up to 3 key-value RDDS together using their keys.   
Another example   
val x = sc.parallelize(List((1, "apple"), (2, "banana"), (3, "orange"), (4, "kiwi")), 2)   
val y = sc.parallelize(List((5, "computer"), (1, "laptop"), (1, "desktop"), (4, "iPad")), 2)   
x.cogroup(y).collect   
Array(Int,(Iterable[String], Iterable[String]))] = Array(   
(4,(ArrayBuffer(kiwi),ArrayBuffer(iPad))),   
(2, (ArrayBuffer(banana),ArrayBuffer())),   
(3,(ArrayBuffer(orange),ArrayBuffer())),   
(1 ,(ArrayBuffer(apple),ArrayBuffer(laptop, desktop))),   
(5, (ArrayBuffer(),ArrayBuffer(computer))))**

**==================================================================================**

**Problem Scenario 52 : You have been given below code snippet.   
val b = sc.parallelize(List(1,2,3,4,5,6,7,8,2,4,2,1,1,1,1,1))  
Operation\_xyz   
Write a correct code snippet for Operation \_xyz which will produce below output  
Scala.collection.Map[Int,Long]= Map(5->1,8->1,3->1,6->1,1->6,2->3,4->2,7->1)   
==================================================================================**

**Solution :   
b.countByValue   
countByValue   
Returns a map that contains all unique values of the RDD and their respective occurrence counts. (Warning: This operation will finally aggregate the information in a single reducer.)  
Listing Variants   
def countByValue(): Map[T, Long]**

**==================================================================================**

**Problem Scenario 53 : You have been given below code snippet.   
val a = sc.parallelize(1 to 10, 3)   
operation1  
b.collect   
Output 1   
Array[lnt] = Array(2,4,6,8,10)   
operation2   
Output 2   
Array[lnt] = Array(1,2,3)   
Write a correct code snippet tor operation1 and operation2 which will produce desired output, shown above.**

**===================================================================================**

**Solution :   
val b = a.filter(\_ % 2 == 0)   
a.filter(\_ < 4).collect   
filter   
Evaluates a boolean function for each data item of the RDD and puts the items for which the function returned true into the resulting RDD.   
When you provide a filter function, it must be able to handle all data items contained in the RDD. Scala provides so-called partial functions to deal with mixed data-types.(Tip:  
Partial functions are very useful if you have some data which may be bad and you do not want to handle but for the good data (matching data) you want to apply some kind of map  
function. The following article is good. It teaches you about partial functions in a very nice way and explains why case has to be used for partial functions:article)**

**Examples for mixed data without partial functions**

**Val b = sc.parallelize(1 to 8)**

**b.filter(\_<4).collect**

**res15:Array[Int] = Array(1,2,3)**

**val a = sc.parallelize(List(“cat”,”horse”,4.0,3.5,2,”dog”))**

**a.filter(\_<4).collect**

**error:value<is not a member of Any**

**=====================================================================================**

**Problem Scenario 54 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"))   
val b = a.map(x => (x.length, x))   
operation 1   
Write a correct code snippet for operation1 which will produce desired output, shown below.   
Array[(Int,String)] = Array((4,lion), (7,panther), (3,dogcat), (5,tigereagle))   
===================================================================================**

**Solution :   
b.foldByKey("")(\_ + \_).collect   
foldBYKey [Pair]  
Very similar to fold, but performs the folding separately for each key of the RDD. This function is only available if the RDD consists ot two-component tuples.   
Listing Variants   
def foldByKey(zeroValue: V)(func:(V,V)=>V):RDD[(K,V)]  
def foldByKey(zeroValue: V, numpartitions: Int)(func: (V, V)=> V):RDD[(K, V)]   
def foldByKey(zeroValue: V, partitioner: Partitioner)(func: (V, V) => V):RDD[(K, V)]**

**====================================================================================**

**Problem Scenario 55 : You have been given below code snippet.   
val pairRDD1 = sc.parallelize(List( ("cat",2), ("cat", 5), ("book", 4),("cat", 12)))   
val pairRDD2 = sc.parallelize(List( ("cat" ,2), ("cup", 5), ("mouse", 4),("cat", 12)))   
operation 1   
Write a correct code snippet for operation1 which will produce desired output, shown below.   
Array[(String, (Option[lnt], Option[lnt]))] = Array((book,(Some(4),None)), (mouse,(None,Some(4))), (cup,(None,Some(5))), (cat,(Some(2),Some(2))), (cat,(Some(2),Some(12))), (cat,(Some(5),Some(2))), (cat,(Some(5),Some(12))), (cat,(Some(112),Some(2))), (cat,(Some(12),Some(12))),**

**================================================================================**

**Solution :   
pairRDD1.fullouter.Join(pairRDD2).collect   
fullOuterJoin [Pair]   
Performs the full outer join between two paired RDDS.   
Listing Variants   
def fullOuterJoin[W](other:RDD[(K,W)],numPartitions:Int):RDD[(K,(Option[V],Option[W]))]  
def fullOuterJoin[W](other:RDD[(K,W)]:RDD[(K,(Option[V],Option[W]))]  
def fullOuterJoin[W](other:RDD[(K,W)],partitioner:partitioner):RDD[(K,(Option[V],Option[W]))**

**Problem Scenario 56 : You have been given below code snippet.   
val a = sc.parallelize(1 to 100, 3)   
operation 1   
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[Array[Int]] = Array(Array(1,2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,26,27,28,29,30,31,32,33),   
Array(34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6O, 61, 62, 63, 64, 65, 66),   
Array(67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, SO, 91, 92,   
94, 95, 96,97, 98, 99, 100))**

**Solution :   
a.glom.collect   
glom   
Assembles an array that contains all elements of the partition and embeds it in an RDD. Each returned array contains the contents of one partition.**

**=====================================================================================**

**Problem Scenario 57 : You have been given below code snippet.   
val a = sc.parallelize(1 to 9, 3)   
operation 1   
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[(String, Seq[lnt])] = Array((even,ArrayBuffer(2, 4, 6, 8)), (odd,ArrayBuffer(1, 3, 5, 7, 9)))**

**Solution :   
a.groupBy(x=> { if (x % 2 == 0) "even" else "odd”}).collect**

**=====================================================================================**

**Problem Scenario 58 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "spider", "eagle"), 2)   
val b = a.keyBy(\_.length)   
operation 1   
Write a correct code snippet for operation1 which will produce desired output, shown below.   
Array[(lnt, Seq[String])] = Array((4,ArrayBuffer(lion)), (6,ArrayBuffer(spider)), (3,ArrayBuffer(dog, cat)), (5,ArrayBuffer(tiger, eagle)))**

**Solution :   
b.groupByKey.collect   
groupByKey [Pair]   
Very similar to groupBy, but instead of supplying a function, the key-component of each pair will automatically be presented to the partitioner.   
Listing Variants   
def groupBYKey():RDD[(K, Iterable[V])]   
def groupByKey(numPartitions: Int):RDD[(K, Iterable[V])]   
def groupByKey(partitioner: Partitioner):RDD[(K, Iterable[V])]**

**=====================================================================================**

**Problem Scenario 59 : You have been given below code snippet.   
val x = sc.parallelize(1 to 20)   
val y = sc.parallelize(10 to 30)   
operation 1   
z.collect   
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[Int] = Array(16, 12, 20, 13, 17, 14, 18, 10, 19, 15, 11)   
Solution :   
val z = x.intersection(y)   
intersection : Retürns the elements in the two RDDS which are the same.**

**===================================================================================**

**Problem Scenario 60 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3)   
val b = a.keyBy(\_.length)   
val c =sc.parallelize(“dog”, "cat", "gnu", "salmon "rabbit", "turkey", "wolf", "bear", " bee"), 3)   
val d = c.keyBy(\_.length)   
operation1  
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[(Int,(String, String))] = Array((6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)), (6,(salmon,salmon)), (6,(salmon,rabbit)), (6,(salmon,turkey)), (3,(dog,dog)), (3,(dog,cat)), (3,(dog,gnu)), (3,(dog,bee)), (3,(rat,dog)), (3,(rat,cat)), (3,(rat,gnu)), (3,(rat,bee)))**

**Solution :**

**b.join(d).collect   
join [Pair] : Performs an inner join using two key-value RDDS. Please note that the keys must be generally comparable to make this work.   
keyBy : Constructs two-component tuples (key-value pairs) by applying a function on each data item. The result of the function becomes the key and the original  
data item becomes the value of the newly created tuples.**

**=====================================================================================**

**Problem Scenario 61 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3)   
val b = a.keyBy(\_.length)   
val c = sc.parallelize(List(“dog”, "cat", "gnu", "salmon", "rabbit", "turkey", "wolf", "bear", "bee"), 3)   
val d = c.keyBy(\_.length)   
operation 1   
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[(Int,(String, Option[String]))] = Array((6,(salmon,Some(salmon))), (6,(salmon,Some(rabbit))), (6,(salmon,Some(turkey))), (6,(salmon,Some(salmon))),   
(6,(salmon,Some(rabbit))), (S,(salmon,some(turkey))), (3,(dog,Some(dog))), (3,(dog,Some(cat))), (3,(dog,Some(gnu))), (3,(dog,Some(bee))), (3,(rat,Some(dog))),  
(3,(rat,Some(cat))), (3,(rat,Some(gnu))), (3,(rat,Some(bee))), (8,(elephant,None)))**

**================================================================================**

**Solution :**

**b.lettOuterJoin(d).collect   
leftOuterJoin [Pair] : Performs an left outer join using two key-value RDDS. Please note that the keys must be generally comparable to make this work correctly  
keyBy : Constructs two-component tuples (key-value pairs) by applying a function on each data item. The result ot the function becomes the key and the original  
data item becomes the value ot the newly created tuples.**

**=================================================================================**

**Problem Scenario 62 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2)   
val b = a.map(x => (x.length, x))   
operationl   
Write a correct code snippet for operation1 which will produce desired output, shown below.   
Array[(Int,String)] = Array((3,xdogx), (5,xtigerx), (4,xlionx), (3,xcatx), (7,xpantherx), (5,xeaglex))**

**================================================================================**

**Solution :**

**b.mapValues("x" + \_ + “X”).collect  
mapValues [Pair] : Takes the values of a RDD that consists of two-component tuples, and applies the provided function to transform each value. Then,it forms new two-component  
tuples using the key and the transformed value and stores them in a new RDD.**

**==================================================================================**

**Problem Scenario 63 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2)   
val b = a.map(x => (x.length, x))   
operation1  
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[(Int,String)] = Array((4,lion), (3,dogcat), (7,panther), (5,tigereagle))**

**Solution :**

**b. reduceByKey(\_ + \_).collect   
reduceByKey [Pair] : This function provides the well-known reduce functionality in Spark. Please note that any function f you provide,   
should be commutative in order to generate reproducible results.**

**=================================================================================**

**Problem Scenario 64 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "salmon", "salmon", "rat", "elephant"), 3)   
val b = a.keyBy(\_.length)   
val c = sc.parallelize(List(“dog” "cat", "gnu", "salmon", "rabbit", "turkey", "wolf", "bear", "bee"), 3)   
val d = c.keyBy(\_.length)   
operation 1   
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[(Int,(Option[String], String))] = Array((6,(Some(salmon),salmon)), (6,(Some(salmon),rabbit)), (6,(Some(salmon),turkey)), (6,(Some(salmon),salmon)),  
(6,(Some(salmon),rabbit)), (6,(Some(salmon),turkey)), (3,(Some(dog),dog)), (3,(Some(dog),cat)), (3,(Some(dog),gnu)), (3,(Some(dog),bee)), (3,(Some(rat),dog))   
(3,(Some(rat),cat)), (3,(Some(rat),gnu)), (3,(Some(rat),bee)), (4,(None,wolt)), (4,(None,bear)))**

**Solution**

**b.rightOuterJoin(d).collect   
rightOuterJoin [Pair] : Performs an right outer join using two key-value RDDS. Please note that the keys must be generally comparable to make this work correctly.**

**===============================================================================**

**Problem Scenario 65 : You have been given below code snippet.   
val a = sc.parallelize(List(“dog”,"cat", "owl ", "gnu", "ant"), 2)   
val b = sc.parallelize(1 to a.count.tolnt, 2)   
val c = a.zip(b)   
operation 1   
Write a correct code snippet tor operation1 which will produce desired output, shown below.   
Array[(String, Int)] = Array((owl,3), (gnu,4), (dog,l), (cat,2), (ant,5))**

**==============================================================================**

**Solution :**

**c. sortByKey(false).collect   
sortByKey [Ordered] : This function sorts the input RDD's data and stores it in a new RDD. The output ROD is a shuffled RDD because it stores data that is output by a reducer  
which has been shuffled. The implementation of this function is actually very clever. First, it uses a range partitioner to partition the data in ranges.Within the shuffled RDD   
Then it sorts these ranges individually with mappartitions using standard sort mechanisms.**

**=====================================================================================**

**Problem Scenario 66 : You have been given below code snippet.   
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "spider", "eagle"), 2)   
val b = a.keyBy(\_.length)   
val c = sc.parallelize(List("ant", "falcon", "squid"), 2)   
val d = c.keyBy(\_.length)   
operation 1   
Write a correct code snippet for operationl which will produce desired output, shown below.   
========================================================================   
Solution :   
b.subtractByKey(d).collect   
subtractByKey [Pair] : Very similar to subtract, but instead of supplying a function, the key-component of each pair will be automatically used as criterion for removing items from the first RDD.**

**==========================================================================**

**Problem Scenario 67 : You have been given below code snippet.   
lines = sc.parallelize([‘its fun to have fun,‘,’but you have to know how.'])   
rl = lines.map( lambda x: x.replace(',',' ').replace('.',' ').replace('-',' ').lower())   
r2 = rl.flatMap(lambda x: x.split())   
r3 = r2.map(lambda x: (x, 1))   
operationl   
r5 = r4.map(lambda x:(x[1].x[0]))   
r6 = r5.sortByKey(ascending=False)   
r6.take(20)   
Write a correct code snippet for operationl which will produce desired output, shown below.   
[(2,'fun'), (2, 'to'), (2, 'have'), (1, 'its'), (1, 'know'), (1, 'how'), (1,'you'), (1, 'but'))**

**==========================================================================  
Solution :   
r4 = r3.reduceByKéY(lambda x,y:x+y)**

**Problem Scenario 68 : You have given a file as below.   
spark75/file1.txt   
File contain some text. As given in RHS (Righ hand side).   
A bigram is pair of successive tokens in some sequence. Please build bigrams from the sequences of words in each sentence,   
and then try to find the most frequently occuring ones.**

**=========================================================================  
Solution :   
Step 1 : Create all three files in hdfs (We will do using Hue). However, you can first create in local filesystem and then upload it to hdfs.   
Step 2 : The first problem is that values in each partition ot our initial RDD describe lines from the file rather than sentences.   
Sentences may be split over multiple lines.   
The glom() RDD method is used to create a single entry for each document containing the list of all lines, we can then join the lines up,   
then resplit them into sentences using “.” as the separator, using flatMap so that every object in our RDD is now a sentence.   
sentences = sc.textFile(“spark75/file1.txt") \   
.glom() \   
.map(lambda x: "”.join(x)) \   
.flatMap(lambda x: x.split(".”))   
Step 3 : Now we have isolated each sentence we can split it into a list of words and extract the word bigrams from it. Our new ROD contains tuples   
containing the word bigram (itself a tuple containing the first and second word) as the first value and the number 1 as the second value.   
bigrams = sentences.map(lambda x:x.split()) \   
.flatMap(lambda x:[((x[i],x[i+1]),1) for i in range(0,len(x)-1)])**

**Step 4 : Finally we can apply the same reduceByKey and sort steps that we used in the wordcount example, to count up the bigrams and sort   
them in order ot descending frequency. In reduceByKey the key is not an individual word but a bigram.   
freq\_bigrams = bigrams.reduceByKey(lambda x,y:x+y) \   
.map(lambda x: (x[1].x[0])) \   
.sortByKey(False)   
freq\_bigrams.take(10)**

**=========================================================================**

**Problem Scenario 69 : Write down a Spark Application using Python,   
In which it read a file "Content.txt" (On hdfs) with following content.   
And filter out the word which is less than 2 characters and ignore all empty lines.   
Once doen store the filtered data in a directory called "problem84" (On hdfs)   
Content.txt   
Hello this is HadoopExam.com   
This is QuickTechie.com   
Apache Spark Training   
This is Spark Learning Session   
Spark is faster than MapReduce**

**===========================================================================**

**Solution :   
Step 1 : Create an application with following code and store it in problem84.py   
# Import SparkContext and SparkConf   
from pyspark import SparkContext, SparkConf   
# Create configuration object and set App name   
conf = SparkConf().setAppName("CCA 175 Problem 84")   
sc = SparkContext(conf=conf)   
#load data from hdfs   
contentRDD = sc.textFile("Content.txt")**

**#filter out non-empty lines   
nonempty\_lines = contentRDD.filter(lambda x: len(x) > O)   
#Split line based on space   
words = nonempty\_lines.flatMap(lambda x: x.split(' '))   
#filter out all 2 letter words   
finalRDD = words.filter(lambda x: len(x) > 2)   
for word in finalRDD.collect():   
print(word)   
#Save final data**

**finalRDD.saveAsTextFile(“Problem84”)   
Stpe 2 : Submit this application   
spark-submit --master yarn problem84.py**

**===================================================================================**

**Problem Scenario 70 : Write down a Spark Application using Python,   
In which it read a file "Content.txt" (On hdfs) with following content.   
Do the word count and save the results in a directory called "problem85" (On hdfs)   
Content.txt   
Hello this is HadoopExam.com   
This is QuickTechie.com   
Apache Spark Training   
This is Spark Learning Session   
Spark is faster than MapReduce**

**======================================================================   
Solution :   
Step 1 : Create an application with following code and store it in problem84.py   
# Import SparkContext and SparkConf   
from pyspark import SparkContext, SparkConf   
# Create configuration object and set App name   
conf = SparkConf().setAppName("CCA 175 Problem 85")   
sc = SparkContext(conf=conf)   
#load data from hdfs   
contentRDD = sc.textFile("Content.txt")**

**#filter out non-empty lines   
nonempty\_lines = contentRDD.filter(lambda x: len(x) > O)   
#Split line based on space   
words = nonempty\_lines.flatMap(lambda x: x.split('’))   
#Do the word count   
wordcounts = words.map(lambda x: (x, 1)) \   
.reduceByKey(lambda x, y: x+y) \   
.map(lambda x: (x[l],x[0])).sortByKey(Flase)   
for word in wordcounts.collect():   
print(word)   
#Save final data**

**Wordcounts.saveAsTextFile(“problem85”)   
Stpe 2 : Submit this application   
spark-submit --master yarn problem85.py**

**Problem Scenario 71 .   
Write down a Spark script using Python,   
In which it read a file "Content.txt" (On hdfs) with following content.   
After that split each row as (key, value), where key is first word in line and entire line as value.   
Filter out the empty lines.   
And save this key value in "problem86" as Sequence file(On hdfs)   
Part 2 : Save as sequence file , where key as null and entire line as value. Read back the stored sequence files.   
Content.txt   
Hello this is HadoopExam.com   
This is QuickTechie.com   
Apache Spark Training   
This is Spark Learning Session   
Spark is faster than MapReduce   
===========================================================================   
Solution :   
Step 1 :   
# Import SparkContext and SparkConf   
from pyspark import SparkContext, SparkConf**

**step 2 :   
#load data from hdfs   
contentRDD = sc.textFile("Content.txt")   
step 3 :   
#filter out non-empty lines   
nonempty\_lines = contentRDD.filter(lambda x: len(x) > O)   
Step 4:   
#Split line based on space (Remember : It is mandatory to convert is in tuple)   
words = nonempty\_lines.map(lambda x: tuple(x.split(' ',1)))   
words.saveAsSequenceFile("problem86")   
Step 5: Check contents in directory problem86   
hdfs dfs -cat problem86/part\*   
Step 6 : Create key, value pair (where key is null)   
nonempty\_lines.map(lambda line: (None, line)).saveAsSequenceFile("problem86\_1")   
Step 7 : Reading back the sequence file data using spark.   
seqRDD = sc.sequenceFile(“problem86\_1")   
Step 8 : Print the content to validate the same.   
for line in seqRDD.collect():   
print(line)**

**Problem Scenario 72 : You have been given a table named "employee2" with following detail.   
first\_name string   
last\_name string   
Write a spark script in python which read this table and print all the rows and individual column values.**

**==========================================================================   
Solution :   
Step 1 : Import statements for HiveContext   
from pyspark.sql import HiveContext   
Step 2 : Create sqlContext   
sqlContext = HiveContext(sc)   
Step 3 : Query hive   
employee2 = sqlContext.sql("select \* from employee2")**

**Step 4 : Now prints the data   
for row in employee2.collect():   
print(row)   
Step 5 : Print specific column   
for row in employee2.collect():   
print(row.first\_name)**

**Problem Scenario 73 : You have been given data in json format as below.   
{"first\_name":"Ankit","last\_name".”jain”},   
{"first\_name":"Amir",”last\_name”.”Khan”}   
{"first\_name":"Rajesh", "last\_name":"Khanna"}   
{"first\_name":"Priynka", "last\_name":"Chopra"}   
{"first\_name":"Kareena", "last\_name":"Kapoor"}   
{"first\_name":"Lokesh",”last\_name”.”Yadav”}   
Do the following activity   
1. create employee.json file locally.   
2. Load this file on hdfs   
3. Register this data as a temp table in Spark using Python.   
4. Write select query and print this data.   
5. Now save back this selected data in json format.**

**===========================================================================**

**Solution :   
Step 1 : create employee.json tile locally.   
vi employee.json   
(press insert)   
past the content.**

**Step2:Upload this file to hdfs,default location**

**Hadoop fs -put employee.json**

**Step3:Write spark script**

**#Import SQLContext**

**From pyspark import SQLContext**

**#Create instance of SQLContext**

**sqlContext=SQLContext(sc)**

**#Load json file**

**Employee=sqlContext.jsonFile(“employee.json”)**

**#Register RDD as a temp table**

**Employee.registerTempTable(“EmployeeTab”)**

**#Select data from Employee table**

**employeeInfo=sqlContext.sql(“select \* from EmployeeTab”)**

**#Iterate data and print**

**For row in employeeInfo.collect():**

**Print(row)**

**Step4:Write data as s Text file**

**employeeInfo.toJSON().saveAsTextFile(“employeeJson1”)**

**Step5:Check whether data has been created or not**

**Hadoop fs -cat employeeJson1/part\***

**Problem Scenario 74 : You have been given MySQL DB with following details.   
User=retail\_dba   
password=cloudera   
database=retail\_db   
table=retail\_db.orders   
Table=retail\_db.order\_items   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Columns of order table : (order\_id , order\_date , order\_customer\_id, order\_status)**

**Columns of order\_items table : (order\_item\_id , order\_item\_order\_   
id , order\_item\_product\_id, order\_item\_quantity,order\_item\_subtotal,order\_**

**item\_product\_price)   
Please accomplish following activities.   
1. Copy "retail\_db.orders" and "retail\_db.order\_items" table to hdfs in respective directory p89\_orders and p89\_order\_items .   
2. Join these data using order\_id in Spark and Python   
3. Now fetch selected columns from joined data Orderld, Order date and amount collected on this order.   
4. Calculate total order placed for each date, and produced the output sorted by date.   
=====================================================================**

**Solution :**

**Step 1 : Import Single table .   
sqoop import –connect jdbc:mysql://quickstart:3306/retail\_db –username=retail\_dba --password=cloudera -table=orders\_items—target-dir=p89\_orders –m1   
sqoop import –connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba --password=cloudera -table=order\_items --target-dir=p89\_order\_items –m1**

**Note : Please check you dont have space between before or after ‘=’ sign.   
Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs   
Step 2 : Read the data from one of the partition, created using above command.   
hadoop fs -cat p89\_orders/part-m-00000   
hadoop fs -cat p89\_order\_items/part-m-OOOOO   
Step 3 : Load these above two directory as RDD using Spark and Python (Open pyspark terminal and do following).   
orders = sc.textFile("p89\_orders")   
orderltems = sc.textFile("p89\_order\_items")   
Step 4 : Convert RDD info key value as (order\_id as a key and rest of the values as a value)   
#First value is order\_id   
ordersKeyValue = orders.map(lambda line: (int(line.split(“,”)[0]), line))   
#Second value as an Order id   
orderltemsKeyValue = orderltems.map(lambda line: (int(line.split(",”)[1]),line))   
Step 5 : Join both the RDD using order\_id   
joinedData = orderltemsKeyValue.join(ordersKeyValue)   
#print the joined data   
for line in joinedData.collect():   
print(line)   
Format of joinedData as below.   
[Orderid, 'All columns from orderltemsKeyValue' , 'All columns from ordersKeyValue']**

**Step 6 : Now fetch selected values Orderld, Order date and amount collected on this order.   
revenuePerOrderPerDay = joinedData.map(lambda row: (row[O],row[1][1].split(“,”)[1],float(row[1][0].split(“,”)[4])))   
#print the result   
for line in revenuePerOrderPerDay.collect():   
print(line)   
Step 7 : Select distinct order ids for each date.   
#distinct(date,order\_id)   
distinctOrdersDate = joinedData.map(lambda row:row[1][1].split(“,”)[1]+”,”+str(row[0])).distinct()   
for line in distinctOrdersDate.collect():**

**print(line)   
Step 8 : Similar to word count , generate (date, 1) record for each row.   
newLineTuple = distinctOrdersDate.map(lambda line: (line.split(",”)[0],1))   
Step 9 : Do the count for each key(date), to get total order per date.   
totalOrdersPerDate = newLineTuple.reduceByKey(lambda a, b: a + b)   
#print results   
for line in totalOrdersPerDate.collect():   
print(line)   
Step 10 : Sort the results by date   
sortedData=totalOrdersPerDate.sortByKey().collect()   
#print results   
for line in sortedData:   
print(line)   
==========================================================================**

**Problem Scenario 75 : You have been given MySQL DB with following details.   
user=retail\_dba   
password=cloudera   
database=retail\_db   
table=retail\_db.order\_items   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Please accomplish following activities.   
1. Copy "retail\_db.order\_items" table to hdfs in respective directory P90\_order\_items.   
2. Do the summation of entire revenue in this table using pyspark.   
3. Find the maximum and minimum revenue as well.   
4. Calculate average revenue**

**Columns of order\_items table : (order\_item\_id , order\_item\_order\_id , order\_item\_product\_id, order\_item\_quantity,order\_item\_subtotal,order\_item\_product\_price)**

**=========================================================================**

**Solution :   
Step 1 : Import Single table .   
sqoop import –connect jdbc:mysql://quickstart:3306/retail\_db –username=retail\_dba --password=cloudera -table=order\_items --target-dir=p90\_order\_items—m1**

**Note : Please check you dont have space between before or after ‘=’ sign.   
Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs   
Step 2 : Read the data from one of the partition, created using above command.   
hadoop fs -cat p90\_order\_items/part-m-00000   
Step 3 : In pyspark, get the total revenue across all days and orders.   
entireTableRDD = sc.textFile("p90\_order\_items")   
#Cast string to float   
extractedRevenueColumn = entireTableRDD.map(lambda line:flat(line.split(“,”)[4]))   
Step 4 : Verity extracted data   
for revenue in extractedRevenueColumn.collect():   
print revenue   
#use reduce function to sum a single column vale   
totalRevenue = extractedRevenueColumn.reduce(lambda a, b: a + b)   
Step 5 : Calculate the maximum revenue   
maximumRevenue = extractedRevenueColumn.reduce(lambda a, b: (a if a>=b else b) )   
Step 6 : Calculate the minimum revenue   
minimumRevenue = extractedRevenueColumn.reduce(lambda a, b: (a if a<=b else b) )**

**Step 7 : Caclculate average revenue**

**Count=extractedRevenuecolumn.count()   
averageRev=totalRevenue/count**

**===========================================================================**

**Problem Scenario 76 : You have been given MySQL DB with following details.   
User=retail\_dba   
password=cloudera   
Database=retail\_db   
Table=retail\_db.orders   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Columns ot order table : (order\_id , order\_date , order\_customer\_id,order\_status)   
Please accomplish following activities.   
1. Copy "retail\_db.orders" table to hdfs in a directory p91\_orders.**

**2. Once data is copied to hdfs, using pyspark calculate the number of order for each status.   
3. use all the following methods to calculate the number of order for each status. (You need to know all these functions and its behavior for real exam)   
- countByKey()   
- groupByKey()   
- reduceByKey()   
- aggregateByKey()   
- combineByKey()   
=========================================================================  
Solution :   
Step 1 : Import Single table   
sqoop import –connect jdbc:mysql://quickstart:3306/retail\_db –username=retail\_dba --password=cloudera -table=orders --target-dir=p91\_orders –m1**

**Note : Please check you dont have space between before or after '=' sign.   
Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs   
Step 2 : Read the data from one of the partition, created using above command.   
hadoop fs -cat p91\_orders/part-m-00000   
Step 3 : countByKey   
#Number of orders by status   
allOrders = sc.textFile("p91\_orders")   
#Generate key and value pairs (key is order status and vale as an empty string   
keyValue = allOrders.map(lambda line: (line.split(",”)[3] “”))   
#Using countByKey, aggregate data based on status as a key   
output=keyValue.countByKey().items()   
for line in output : print(line)   
Step 4 : groupByKey   
#Generate key and value pairs (key is order status and vale as an one   
keyValue = allOrders.map(lambda line: (line.split)”,”)[3],1))   
#Using countByKey, aggregate data based on status as a key   
output= keyValue.groupByKey().map(lambda kv: (kv[O], sum(kv[l])))   
for line in output.collect() : print(line)   
Step 5 : reduceByKey   
#Generate key and value pairs (key is order status and vale as an one   
keyValue = allOrders.map(lambda line: (line.split(",”)[3],1))**

**#Using countByKey, aggregate data based on status as a key   
output= keyValue.reduceByKey(lambda a, b: a + b)   
for line in output.collect() : print(line)**

**Step 6 : aggregateByKey   
#Generate key and value pairs (key is order status and vale as an one**

**keyValue = allOrders.map(lambda line: (line.split(“,”)[3],line))   
output=keyValue.aggregateByKey(0, lambda a, b: a+l, lambda a, b: a+b)   
for line in output.collect() : print(line)   
Step 7 : combineByKey   
#Generate key and value pairs (key is order status and vale as an one   
keyValue = allOrders.map(lambda line: (line.split(“,”)[3], line))   
output=keyValue.combineByKey(lambda value: 1, lambda acc, value: acc+l, lambda acc, value: acc+value)   
for line in output.collect() : print(line)   
#Watch Spark Protessional Training provided by www.HadoopExam.com to understand more on each above functions.   
(These are very important functions for real exam)**

**======================================================================**

**Problem Scenario 77 : You have been given MySQL DB with following details.   
User=retail\_dba   
password=cloudera   
database=retail\_db   
table=retail\_db.orders   
table=retail\_db.order\_items   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Columns of order table : (order\_id , order\_date , order\_customer\_id, order\_status)   
Columns ot order\_items table : (order\_item\_id , order\_item\_order\_   
id , order\_item\_product\_id, order\_item\_quantity,order\_item\_subtotal,order\_item\_product\_price)   
Please accomplish following activities.   
1. Copy "retail\_db.orders" and "retail\_db.order\_items" table to hdfs in respective directory p92\_orders and p92\_order\_items .   
2. Join these data using order\_id in Spark and Python   
3. Calculate total revenue perday and per order   
4. Calculate total and average revenue for each date.   
- combineByKey   
- aggregateByKey**

**======================================================================**

**Solution :**

**Step 1 : Import Single table .**

**sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba --password=cloudera -table=orders\_items --target-dir=p92\_orders --m1   
sqoop import —connect jdbc:mysql://quickstart:3306/retail\_db –username=retail\_dba —password=cloudera -table=order\_items --target-dir=p92\_order\_items –m1   
Note : Please check you dont have space between before or after '=' sign.   
Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs   
Step 2 : Read the data from one of the partition, created using above command.   
hadoop fs -cat p92\_orders/part-m-00000   
hadoop fs -cat p92\_order\_items/part-m-00000   
Step 3 : Load these above two directory as RDD using Spark and Python (Open pyspark terminal and do following).   
orders = sc.textFile("p92\_orders")   
orderltems = sc.textFile("p92\_order\_items")   
Step 4 : Convert RDD into key value as (order\_id as a key and rest of the values as a value)   
#First value is order\_id**

**ordersKeyValue = orders.map(lambda line: (int(line.split(“,”)[1]),line))   
#Second value as an Order\_id   
orderltemsKeyValue = orderltems.map(lambda line: (int(line.split(",”)[1]),line))   
Step 5 : Join both the RDD using order\_id   
joinedData = orderltemsKeyValue.join(ordersKeyValue)   
#print the joined data   
for line in joinedData.collect():   
print(line)**

**Format of joinedData as below.   
[Orderld, 'All columns trom orderltemsKeyValue' , 'All columns from ordersKeyValue']   
Step 6 : Now fetch selected values Orderld, Order date and amount collected on this order.   
//Retruned row will contain ((order\_date,order\_id),amout\_collected)   
revenuePerDayPerOrder = joinedData.map(lambda row: ((row[1][1].split(“,”)[1],row[0]),float(row[1][0].split(“,”)[4])))   
#print the result   
for line in revenuePerDayPerOrder.collect():   
print(line)   
Step 7 : Now calculate total revenue perday and per order   
A. using reduceByKey   
totalRevenuePerDayPerOrder = revenuePerDayPerOrder.reduceByKey(lambda runningSum, value: runningSum + value)   
for line in totalRevenuePerDayPerOrder.sortByKey().collect():   
print(line)**

**#Generate data as(date,amount\_collected)(Ignore order\_Id)**

**dateAndRevenueTuple = totalRevenuePerDayPerOrder.map(lambdaline: (line[0][0],line[1]))**

**for line in dateAndRevenueTuple.sortByKey().collect():**

**print(line)   
Step 8 : Calculate total amount collected for each day. And also calculate number of days.   
#Generate output as (Date, Total Revenue for date, total\_number\_of\_dates)**

**#Line 1 : it will generate tuple (revenue,l)   
#Line 2 : Here, we will do summation for all revenues at the same time another counter to maintain number of records.   
#Line 3 : Final function to merge all the combiner   
totalRevenueAndTotalCount = dateAndRevenueTuple.combineByKey( \   
lambda revenue: (revenue, 1), \   
lambda revenueSumTuple, amount: (revenueSumTuple[O] + amount, revenueSumTuple[1] + 1), \   
lambda tuplel, tuple2: (round(tuple1[0]+ tuple2[O], 2), tuplel[l] + tuple2[1]) \**

**)   
for line in totalRevenueAndTotalCount.collect():   
print(line)   
Step 9 : Now calculate average for each date   
averageRevenuePerDate = totalRevenueAndTotalCount.map(lambda threeElements: (threeElements[O],threeElements[1][0]/threeElements[1][1]))   
for line in averageRevenuePerDate.collect():   
print(line)   
Step 10 : using aggregateByKey   
#line 1 : (Initialize both the value, revenue and count)   
#line 2 : runningRevenueSumTuple (Its a tuple for total revenue and total record count for each date)   
#line 3 : Summing all partitions revenue and count   
totalRevenueAndTotalCount = dateAndRevenueTuple.aggregateByKey( \**

**(0,0),\**

**lambda runningRevenueSumTuple, revenue: (runningRevenueSumTuple[O] + revenue, runningRevenueSumTuple[1] + 1), \   
lambda tupleOneRevenueAndCount, tupleTwoRevenueAndCount: (tupleOneRevenueAndCount[O] + tupleTwoRevenueAndCount[O], tupleOneRevenueAndCount[1]+tupleTwoRevenueAndCount[1])\**

**)**

**For line in totalRevenueAndTotalCount.collect():**

**Print(line)**

**Step 11 : Calculate the average revenue per date   
averageRevenuePerDate = totalRevenueAndTotalCount.map(lambda threeElements: (threeElements[O],threeElements[1][0]/threeElements[1][1]))   
for line in averageRevenuePerDate.collect():   
print(line)**

**=====================================================================**

**Problem Scenario 78 : You have been given MySQL DB with following details.   
User=retail\_dba   
password=cloudera   
Database=retail\_db   
table=retail\_db.orders   
table=retail\_db.order\_items   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Columns of order table : (order\_id , order\_date , order\_customer\_id, order\_status)   
Columns of order\_items table : (order\_item\_id , order\_item\_order\_id , order\_item\_product\_id, order\_item\_quantity,order\_item\_subtotal,order\_item\_product\_price)  
Please accomplish following activities.   
1. Copy "retail\_db.orders" and "retail\_db.order\_items" table to hdfs in respective directory p92\_orders and p92\_order\_items .   
2. Join these data using order\_id in Spark and Python   
3. Calculate total revenue perday and per customer   
4. Calculate maximum revenue customer   
=======================================================================**

**Solution :**

**Step 1 : Import Single table .   
sqoop import –connect jdbc:mysql://quickstart:3306/retail\_db –username=retail\_dba --password=cloudera -table=orders –target-dir=p92\_orders –m1   
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db --username=retail\_dba --password=cloudera -table=order\_items --target-dir=p92\_order\_items –m1**

**Note:Please check you don’t have space between before or after ‘=’sign.**

**Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs**

**Step2: Read the data from one of the partition,created using above command.**

**hadoop fs -cat p92\_orders/part-m-00000**

**Hadoop fs -cat p92\_order\_items/part-m-00000**

**Step3:Load these above two directory as RDD using Spark and Python(Open pyspark terminal and do following).**

**Orders = sc.textFile(“p92\_orders”)**

**Orderitems = sc.textFile(“p92\_order\_items”)**

**Step4: Convert RDD into key value as(oder\_id as a key and rest of the values as a value)**

**#First value is order\_id**

**orderitemsKeyValue = orders.map(lambda line: (int(line.split(“,”)[1]),line))**

**#second value as an Order\_id**

**orderitemsKeyValue=orderitems.map(lambda line: (int(line.split(“,”)[1]),line))**

**Step 5:join both the RDD using order\_id**

**joinedData = orderItemsKeyValue.join(ordersKeyValue)**

**#print the joined data**

**For line in joinedData.collect():**

**Print(line)**

**#Format of joined data**

**#[Orderid, ‘All columns from orderitemsKeyValue’, ‘All columns from ordersKeyValue’]**

**ordersPerDatePerCustomer = joinedData.map(lambda line: ((line[1][1].split(“,”)[1],line[1][1].split(“,”)[2]),float(line[1][0].split(“,”)[4])))**

**amount collectedperdaypercustomer = ordersperdatepercustomer.reducedbykey(lambda runningsum, amount : runningsum + amount)**

**#(outrecord format will be ((dat,customer\_id), totalamount))**

**For line in amountcollectedperdaypercustomer.collect():**

**Print(line)**

**#now change the format of record as(date,(customer\_id,total\_amount))**

**revenueperdatepercustomerRDD=amountcollectedperdaypercustomer.map(lambda threelmenttuple: (threeelementuple[0][0], (threeelementtuple[0][1], threeelementtuple[1])))**

**for line in revenueperdatepercustomerrdd.collect():**

**print(line)**

**#calculate maximum amount collected by a customer for each day**

**Perdatemaxamountcollectedbycustomer=revenueperdatepercustomerrdd.reducedbykey(lambdarunningamounttuple,newamounttuple: (runningamounttuple[1]>=newamounttuple[1] else newamounttuple))**

**For line in perdatemaxamountcollectedbycustomer.sortbykey().collect():**

**Print(line)**

**=========================================================================**

**Problem Scenario 79 : You have been given MySQL DB with following details.   
User=retail\_dba   
password=cloudera   
database=retail\_db   
table=retail\_db.products   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Columns of products table : (product\_id | product\_category\_id | product\_name | product\_description | product\_price | product\_image )   
Please accomplish following activities.   
1. Copy "retail\_db.products" table to hdfs in a directory p93\_products   
2. Filter out all the empty prices   
3. Sort all the products based on price in both ascending as well as descending order.   
4. Sort all the products based on price as well as product\_id in descending order.   
5. use the below functions to do data ordering or ranking and fetch top 10 elements   
top()   
takeordered()   
sortByKey()**

**Solution :**

**Step 1 : Import Single table .**

**sqoop import –connect jdbc:mysql://quickstart:3306/retail\_dba –username=retail dba —password=cloudera -table=products —target-dir=p93\_products –m1**

**Note : Please check you dont have space between before or after'=' sign.   
Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs**

**Step 2 : Step 2 : Read the data from one of the partition, created using above command.   
hadoop fs -cat p93\_products/part-m-00000  
Step 3 : Load this directory as RDD using Spark and Python (Open pyspark terminal and do following).   
productsRDD = sc.textFile("p93\_products")   
Step 4 : Filter empty prices, it exists   
#filter out empty prices lines   
nonempty\_lines = productsRDD.filter(lambda x: len(x.split(2,”)[4])> O)   
Step 5 : Now sort data based on product\_price in order.   
sortedPriceProducts=nonempty\_lines.map(lambda line : (float(line.split(“,”)[4]),line.split(“,”)[2])).sortByKey()   
for line in sortedPriceProducts.collect():   
print(line)   
Step 6 : Now sort data based on product\_price in descending order.   
sortedPriceProducts=nonempty\_lines.map(lambda line : (float(line.split(“,”)[4]),line.split(“,”)[2])).sortByKey(False)   
for line in sortedPriceProducts.collect():   
print(line)   
Step 7 : Get highest price products name.   
sortedPriceProducts=nonempty\_lines.map(lambda line : (float(line.split(“,”)[4]),line.split(“,”)[2])).sortByKey(False).take(1)   
print(sortedPriceProducts)**

**Step 8 : Now sort data based on product\_price as well as product\_id in descending order.   
#Dont forget to cast string   
#Tuple as key ((price,id),name)   
sortedPriceProducts=nonempty\_lines.map(lambda line : ((float(line.split(“,”)[4]),int(line.split(“,”)[0])),line.split(“,”)[2])).sortByKey(False).take(10)  
print(sortedPriceProducts)   
Step 9 : Now sort data based on product\_price as well as product\_id in descending order, using top() function.   
#Dont forget to cast string   
#Tuple as key ((price,id),name)   
sortedPriceProducts=nonempty\_lines.map(lambda line : (float(line.split(“,”)[4]),int(line.split(“,”)[0]))line.split(“,”)[2])).top(10)  
print(sortedPriceProducts)   
Step 10 : Now sort data based on product\_price as ascending and product\_id in ascending order, using takeordered() function.   
#Dont forget to cast string   
#Tuple as key ((price,id),name)   
sortedPriceProducts=nonempty\_lines.map(lambda line : (float(line.split(“,”)[4]),int(line.split(“,”)[0]))line.split(“,”)[2])).takeOrdered(10, lambda tuple : (tuple[0][0],tuple[0][1]))   
Step 11 : Now sort data based on product\_price as descending and product\_id in ascending order, using takeordered() function.   
#Dont forget to cast string   
#Tuple as key ((price,id),name)   
#Using minus(-) parameter can help you to make descending ordering , only for numeric value.   
sortedPriceProducts=nonempty\_lines.map(lambda line : (float(line.split(“,”)[4]),int(line.split(“,”)[0])),line.split(“,”)[2])).takeOrdered(10, lambda tuple: (-tuple[0][0],tuple[0][1]))**

**Problem Scenario 80 : You have been given MySQL DB with following details.**

**User=retail\_dba   
password=cloudera   
database=retail\_db   
table=retail\_db.products   
jdbc URL = jdbc:mysql://quickstart:3306/retail\_db   
Columns of products table : (product\_id | product\_category\_   
id | product\_name | product\_description | product\_price I product\_image )   
Please accomplish following activities.   
1. Copy "retail\_db.products" table to hdfs in a directory p93\_products   
2. Now sort the products data sorted by product price per category, use product\_category\_id colunm to group by category**

**=======================================================================   
Solution :   
Step 1 : Import Single table .   
sqoop import --connect jdbc:mysql://quickstart:3306/retail\_db –username=retai\_dba--password=cloudera -table=products --target-dir=p93\_products –m1   
Note : Please check you dont have space between before or after ‘=’sign.   
Sqoop uses the MapReduce framework to copy data from RDBMS to hdfs   
Step 2 : Step 2 : Read the data from one of the partition, created using above command.   
hadoop fs -cat p93\_products/part-m-00000**

**Step 3 : Load this directory as RDD using Spark and Python (Open pyspark terminal and do following).   
productsRDD = sc.textFile("p93\_products")   
Step 4 : Filter empty prices, if exists   
#filter out empty prices lines   
nonempty\_lines = productsRDD.filter(lambda x:len(x.split(“,”)[4])>0)   
Step 5 : Create data set like (categroyld, (id,name,price)   
mappedRDD = nonempty\_lines.map(lambda line: (line.split(",”)[1],(line.split(“,”)[2],float(line.split(“,”)[4])))   
for line in mappedRDD.collect(): print(line)   
Step 6 : Now groupBy the all records based on categoryld, which a key on mappedRDD it will produce output like (categoryld, iterable of all lines**

**For akey/categoryId)   
groupByCategroyld = mappedRDD.groupByKey()   
for line in groupByCategroyid.collect(): print(line)   
Step 7 : Now sort the data in each category based on price in ascending order.   
# sorted is a function to sort an iterable, we can also specify, what would be the key on which we want to sort in this case we have price on which it needs to be sorted.   
groupByCategroyid.map(lambda tuple: sorted(tuple[l], key=lambda tupleValue: tupleValue[2])).take(5)   
Step 8 : Now sort the data in each category based on price in descending order.   
# sorted is a function to sort an iterable, we can also specify, what would be the key on which we want to sort in this case we have price on which it needs to be sorted.   
groupByCategroyid.map(lambda tuple: sorted(tuple[l], key=lambda tupleValue: tupleValue[2] , reverse=True)).take(5)**

**==========================================================================**

**Problem Scenario 81 : You have been given MySQL DB with following details.   
You have been given following product.csv file   
product.csv (Create this file in hdfs)   
productlD,productCode,name,quantity,price   
1001,PEN,pen Red,5000,1,23**

**1002,PEN,pen Blue,8000,1,25**

**1003,PEN,pen Black,8000,1,25**

**1004,PEC,Pencil 2B,10000,0.48**

**1005,PEC,Pencil 2H,8000,0.49**

**1004,PEC,Pencil HB,0,9999.99  
Now accomplish following activities.   
1.Create a Hive ORC table using SparkSql   
2. Load this data in Hive table.   
3. Create a Hive parquet table using SparkSQL and load data in it.**

**===================================================================**

**Solution :**

**Step 1 : Create this file in HDFS under following directory (Without header)/user/cloudera/he/exam/task1/product.csv   
Step 2 : Now using Spark-shell read the file as RDD**

**//load the data into a new ROD   
val products = sc.textFile("/user/cloudera/he/exam/task1/product.csv")   
//Return the first element in this ROD   
products.first()   
Step 3 : Now define the schema using a case class   
case class Product(productid: Integer, code: String, name: String, quantity:lnteger , price: Float)   
Step 4 : create an RDD of Product objects   
val prdRDD = Product.map(\_.split(“,”)).map(p =>product(p(0).toint,p(1),p(2),p(3).toint,p(4).toFloat))   
prdRDD.first()   
prdRDD.count()   
Step 5 : Now create data frame   
val prdDF = prdRDD.tODF()   
Step 6 : Now store data in hive warehouse directory. (However, table will not be created )   
import org.apache.spark.sql.SaveMode   
prdDF.write.mode(SaveMode.Overwrite).format("orc").saveAsTable("product\_orc\_table")   
Step 7 : Now create table using data stored in warehouse directory. With the help ot hive.   
hive   
show tables   
CREATE EXTERNAL TABLE products (productid int,code string,name string ,quantity int, price float )   
STORED AS orc   
LOCATION '/user/hive/warehouse/product\_orc\_table';   
Step 8 : Now create a parquet table   
import org.apache.spark.sql.SaveMode   
prdDF.write.mode(SaveMode.Overwrite).format("parquet").saveAsTable("product\_parquet\_table")**

**Step 9 : Now create table using this   
CREATE EXTERNAL TABLE products\_parquet (productid int,code string,name string ,quantity int, price float )   
STORED AS parquet   
LOCATION 'luser/hive/warehouse/product\_parquet\_table';   
Step 10 : Check data has been loaded or not.   
select \* from products;   
select \* from products\_parquet;**

**=================================================================================**

**Problem Scenario 82 : You have been given table in Hive with following structure (Which you have created in previous exercise).   
productid int   
code string   
name string   
quantity int   
price float   
using SparkSQL accomplish following activities.   
1. Select all the products name and quantity having quantity <=2000   
2. Select name and price ot the product having code as 'PEN'   
3. Select all the products, which name starts with PENCIL   
4. Select all products which "name" begins with 'P', followed by any two characters, followed by space, followed by zero or more characters**

**Solution .   
Step 1 : Copy following tile (Mandatory Step in Cloudera QuickVM) if you have not done it.   
sudo su root   
cp /usr/lib/hive/cont/hive-site.xml /usr/lib/spark/conf/**

**Step 2 : Now start spark-shell   
Step 3 : Select all the products name and quantity having quantity<=2000   
val results = sqlContext.sql( """SELECT name, quantity FROM products WHERE quantity 2000 """)  
results.show()   
Step 4 : Select name and price of the product having code as 'PEN'   
val results = sqcontext.sql( """ SELECT name, price FROM products WHERE code = 'PEN' """)   
results.show()   
Step 5 : Select all the products , which name starts with PENCIL   
val results = sqcontext.sql(""" SELECT name, price FROM products WHERE upper(name) LIKE 'PENCIL%' """)   
results.show()   
Step 6 : select all products which "name" begins with 'P', followed by any two characters, followed by space, followed by zero or more characters  
--"name" begins with 'P', followed by any two characters,   
--followed by space, followed by zero or more characters   
val results = sqcontext.sql( """ SELECT name, price FROM products WHERE name LIKE 'P\_ %’ """)   
results.show()**

**Problem Scenario 83 : In Continuation of previous question, please accomplish following activities.   
1. Select all the records with quantity >= 5000 and name starts with 'Pen'   
2. Select all the records with quantity >= 5000, price is less than 1.24 and name starts with 'Pen'   
3. Select all the records witch does not have quantity >= 5000 and name does not starts with 'Pen'   
4. Select all the products which name is 'Pen Red', 'Pen Black'   
5. select all the products WhiCh has price BETWEEN 1.0 AND 2.0 AND quantity BETWEEN 1000 AND 2000.**

**Solution :   
Step 1 : Select all the records with quantity >= 5000 and name starts with 'Pen'   
val results = sqcontext.sql( SELECT\* FROM products WHERE quantity >= 5000 AND name LIKE •pen%’ """)   
results.show()   
Step 2 : Select all the records with quantity >=5000 , price is less than 1.24 and name starts with 'Pen'   
val results = sqcontext.sql( "'"'SELECT\* FROM products WHERE quantity >= 5000 AND price < 1.24 AND name LIKE •pen%’ """)  
results.show()   
Step 3 : Select all the records witch does not have quantity >= 5000 and name does not starts with 'Pen'   
val results = sqcontext.sql(""" SELECT\* FROM products WHERE NOT (quantity >=5000 AND name LIKE •pen %)""")   
results.show()**

**Step 4 : Select all the products wchich name is 'Pen Red', 'Pen Black'   
val results = sqcontext.sql( """SELECT\* FROM products WHERE name IN (‘pen Red', 'Pen Black') """ )   
results.show()   
step 5 : select all the products WhiCh has price BETWEEN 1.0 AND 2.0 AND quantity BETWEEN 1000 AND 2000.   
val results = sqcontext.sql(""" SELECT\* FROM products WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000) """)   
results.show()**

**Problem Scenario 84 : In Continuation of previous question, please accomplish following activities.   
1. Select all the products which has product code as null   
2. Select all the products, whose name starts with Pen and results should be order by Price descending order.   
3. Select all the products, whose name starts with Pen and results should be order by Price descending order and quantity ascending order.   
4. Select top 2 products by price   
===========================================================================**

**Solution :   
Step 1 : Select all the products which has product code as null   
val results = sqlContext.sql(“””SELECT \* FROM products WHERE code IS NULL”””)   
results.show()   
val results = sqlContext.sql(“””SELECT \* FROM products WHERE code = NULL”””)   
results.show()   
Step 2 : Select all the products , whose name starts with Pen and results should be order by Price descending order.   
Val results = sqlContext.sql(“””SELECT\* FROM products WHERE name LIKE 'Pen%’ ORDER BY price DESC””” )**

**Results.show()   
Step 3 : Select all the products , whose name starts with Pen and results should be order by Price descending order and quantity ascending order  
Val results = sqlContext.sql(“””SELECT\* FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity””” )**

**results. show()   
Step 4 : Select top 2 products by price   
val results = sqlContext.sql( """SELECT \* FROM products ORDER BY price desc LIMIT 2”””)   
results.show()   
===========================================================================**

**Problem Scenario 85 : In Continuation of previous question, please accomplish following activities.   
1. Select all the columns from product table with output header as below.   
productiD AS ID   
code AS Code   
name AS Description   
price AS ‘unit Price'   
2. Select code and name both separated by ' - ' and header name should be ‘product Description'.   
3. Select all distinct prices.   
4. Select distinct price and name combination.   
5. Select all price data sorted by both code and productlD combination.**

**6. count number of products.   
7. Count number of products for each code.   
===================================================================   
Solution :   
Step 1 : Select all the columns from product table with output header as below.   
productiD AS ID   
code AS Code   
name AS Description   
price AS ‘unit Price'**

**val results = sqlcontext.sql( '"'"SELECT productiD AS ID, code AS code, name AS Description, price AS ‘unit price FROM products ORDER BY ID”””)   
results.show()   
Step 2 : Select code and name both separated by ' - ' and header name should be ‘product Description'.   
val results = sqlcontext.sql( """ SELECT CONCAT(COde   
, '-', name) AS ‘product Description', price FROM products'""' )   
results.show()   
Step 3 : Select all distinct prices.   
val results = sqlcontext.sql( '"'"SELECT DISTINCT price AS 'Distinct price’ FROM products"'"' )   
results.show()   
Step 4 : Select distinct price and name combination.   
val results = sqlcontext.sql( """ SELECT DISTINCT price, name FROM products""" )   
results.show()   
Step 5 : Select all price data sorted by both code and productlD combination.   
val results = sqlcontext.sql( """SELECT\* FROM products ORDER BY code, productiD"'"')   
results.show()   
Step 6 : count number of products.   
val results = sqlcontext.sql(“”” SELECT COUNT(‘) AS ‘count' FROM products""" )   
results.show()   
Step 7 : Count number of products for each code.   
val results = sqlContext.sql('"'"SELECT code,COUNT(‘) FROM products GROUP BY code ORDER BY count DESC"'"')   
results.show()   
val results = sqlContext.sql('"'"SELECT code,COUNT(‘) AS count FROM products GROUP BY code ORDER BY count DESC'""' )**

**results.show()**

**Problem Scenario 86 : In Continuation ot previous question, please accomplish following activities.   
1.Select Maximum, minimum, average , Standard Deviation, and total quantity.   
2. Select minimum and maximum price for each product code.   
3. Select Maximum, minimum, average , Standard Deviation, and total quantity for each product code, hwoever make sure Average and Standard deviation will have maximum two decimal values.   
4. Select all the product code and average price only where product count is more than or equal to 3.   
5. Select maximum, minimum , average and total of all the products for each code. Also produce the same across all the products.**

**Solution :   
Step 1 : Select Maximum, minimum, average , Standard Deviation, and total quantity.   
val results = sqcontext.sql( “""SELECT MAX(price) AS MAX , MIN(price) AS**

**MIN , AVG(price) AS Average, STD(price) AS STD, SUM(quantitY) AS total\_products From products””” )  
results. show()**

**Step 2 : Select minimum and maximum price tor each product code.   
val results = sqlContext.sql( """ SELECT code, MAX(price) AS •Highest Price', MIN(price) AS •Lowest Price'   
FROM products   
GROUP BY code""")**

**results.show()   
Step 3 : Select Maximum, minimum, average , Standard Deviation, and total quantity for each product code, hwoever make sure Average and stand deviation will have maximum two decimal values.   
val results = sqlContext.sql( """ SELECT code, MAX(price), MIN(price),   
CAST(AVG(price) As DECIMAL(7,2)) AS ‘AVERAGE’,   
CAST(STD(price) As DECIMAL(7,2)) AS ‘Std Dev’,  
SUM(quantity)   
FROM products**

**GROUP BY code """)   
results.show()   
Step 4 : Select all the product code and average price only where product count is more than or equal to 3.   
val results = sqlContext.sql( """SELECT code AS •product Code' ,   
COUNT(\*)AS •count',   
CAST(AVG(Price)AS DECIMAL(7,2))AS 'Average   
FROM products   
GROUP BY code  
HAVING count >=3""" )   
results. show()**

**Step 5 : Select maximum, minimum , average and total ot all the products tor each code. Also produce the same across all the products.   
val results = sqlContext.sql( """ SELECT   
code,   
MAX(price),   
MIN(price),   
CAST(AVG(PRICE) AS DECIMAL(7,2)AS'Average ,   
SUM(quantity)   
FROM products   
GROUP BY code   
WITH ROLLUP'""' )   
results. show()**

**==================================================================================**

**Problem Scenario 87 : You have been given below three Files   
product.csv (Create this File in hdfs)   
productlD,productCode,name,quantity,price,supplierid**

**1001,PEN,pen Red,5000,1.23,501   
1002,PEN,pen blue,8000,1.25,501**

**1003,PEN,pen Black,2000,1.25,501**

**1004,PEC,pencil 2B,10000,0.48,502  
1005,PEC,pencil 2H,8000,0.49,502**

**1006,PEC,pencil 2B,10000,0.48,502  
2001,PEC,pencil 3B,5000,0.52,501**

**2002,PEC,pencil 4B,200,0.62,501  
2003,PEC,pencil 5B,100,0.73,501**

**2004,PEC,pencil 6B,500,0.47,502**

**Supplier.csv**

**Supplierid,name,phone   
501,ABC Traders,88881111**

**502,XYZ company,8882222**

**503,QQ corp,88883333**

**products\_suppliers.csv**

**productlD,supplierID**

**2001,501**

**2002, 501**

**2003, 501**

**2004, 502**

**2001,503**

**Now accomplish all the queries given in solution.**

**Select product, its price , its supplier name where product price is less than 0.6 using SparkSQL   
Solution •**

**Step 1 :   
hdfs dfs -mkdir sparksq12   
hdfs dfs -put product.csv sparksq12/   
hdfs dfs -put supplier.csv sparksq12/   
hdfs dfs -put products\_suppliers.csv sparksq12/   
Step 2 : Now in spark shell   
//this is used to implicitly convert an ROD to a DataFrame.   
import sqlContext.implicits.   
//Import Spark SQL data types and Row.   
import org.apache.spark.sql.**

**//load the data into a new RDD   
val products = sc.textFile("sparksql2/product.csv")   
val supplier = sc.textFile("sparksq12/supplier.csv")   
val prdsup = sc.textFile("sparksq12/products\_suppliers.csv")   
//Return the first element in this RRD   
products.first()**

**supplier.first()   
prdsup.first()   
//define the schema using a case class   
case class Product(productid: Integer, code: String, name: String, quantity:lnteger , price: Float , supplierid:lnteger)   
case class Suplier(supplierid: Integer, name: String, phone: String)   
case class PRDSUP(productid: Integer,supplierid: Integer)   
//create an RRD ot Product objects   
val prdRDD = products.map(\_.split(",")).map(p => Product(p(O).tolnt,p(1),p(2),p(3).tolnt,p(4).toFloat,p(5).tolnt ))   
val supRDD = supplier.map(\_.split(",")).map(p => Suplier(p(O).tolnt,p(1),p(2)))   
val prdsupRDD = prdsup.map(\_.split(",")).map(p => PRDSUP(P(0).TOINT,P(1).tolnt))  
prdRDD.tirst()   
prdRDD.count()   
supRDD.first()   
supRDD.count()   
prdsupRDD.nrst()   
prdsupRDD.count()   
//change RDD of product objects to a DataFrame   
val prdDF = prdRDD.tODF()   
val supDF = supRDD.tODF()   
val prdsupDF = prdsupRDD.tODF()**

**//register the DataFrame as a temp table   
prdDF.registerTempTable("products")   
supDF.registerTempTable("suppliers")   
prdsupDF.registerTempTable("products\_suppliers")   
//Select product, its price , its supplier name where product price is less than 0.6   
val results = sqlContext.sql( """ SELECT products.name, price, suppliers.name as sup\_name FROM products JOIN suppliers ON products.supplierID = suppliers.supplierID WHERE PRICE <0.6”””)  
results.show()**

**Problem Scenario 88 : You have been given below three Files   
product.csv (Create this File in hdfs)   
productlD,productCode,name,quantity,price,supplierid   
1001,PEN,pen Red,5000,1.23,501   
1002,PEN,pen blue,8000,1.25,501**

**1003,PEN,pen Black,2000,1.25,501**

**1004,PEC,pencil 2B,10000,0.48,502  
1005,PEC,pencil 2H,8000,0.49,502**

**1006,PEC,pencil 2B,10000,0.48,502  
2001,PEC,pencil 3B,5000,0.52,501**

**2002,PEC,pencil 4B,200,0.62,501  
2003,PEC,pencil 5B,100,0.73,501**

**2004,PEC,pencil 6B,500,0.47,502**

**supplier.csv   
supplierid,name,phone   
501,XYZ company,88882222   
503,QQ corp,88883333   
products\_suppliers.csv   
productlD,supplierlD\_   
2001,501   
2002, 501   
2003, 501   
2004, 502   
2001 ,503   
Now accomplish all the queries given in solution.**

**1. It is possible that , same product can be supplied by multiple supplier. Now tind each product , its price according to each supplier.   
2. Find all the supllier name, who are supplying 'Pencil 3B'   
3. Find all the products , which are supplied by ABC Traders.**

**Solution :**

**Step 1 : It is possible that , same product can be supplied by multiple supplier. Now tind each product , its price according to each supplier.   
val results = sqlContext.sql( """ SELECT products.name AS •product Name', price, suppliers.name AS •supplier Name'   
FROM products \_ suppliers   
JOIN products ON products \_ suppliers.productlD = products.productlD   
JOIN suppliers ON products\_suppliers.supplierlD = suppliers.supplierlD   
results. show()**

**Step 2 : Find all the supllier name, who are supplying 'Pencil 3B'   
val results = sqlContext.sql( """ SELECT p.name AS •product Name', s.name AS •supplier Name'**

**FROM products \_ suppliers AS ps   
JOIN products AS p ON ps.productlD = p.productlD   
JOIN suppliers AS s ON ps.supplierlD = s.supplierlD   
WHERE p.name = 'Pencil 3B'""" )   
results.show()**

**Step 3 : Find all the products , which are supplied by ABC Traders.   
val results = sqlContext.sql( """SELECT p.name AS •product Name', s.name AS •supplier Name'   
FROM products AS p, products \_ suppliers AS ps, suppliers AS s   
WHERE p.pr0ductlD = ps.pr0ductlD   
AND ps.supplierlD = s.supplierlD   
AND s.name = 'ABC Traders"""' )**

**results.show()**

**Problem Scenario 89 : You have been given below patient data in csv format.   
patients.csv   
patientlD,name,dateOtBirth,lastVisitDate   
1001,Ah Teck,1991-12-31,2012-01-20  
1002, Kumar, 2011-10-29,2012-09-20   
1003,Ali,2011-01-30,2012-10-21**

**Accomplish following activities.   
1. Find all the patients whose lastVisitDate between current time and '2012-09-15'   
2. Find all the patients who born in 2011   
3. Find all the patients age   
4. List patients whose last visited more than 60 days ago   
5. Select patients 18 years old or younger   
Solution :**

**Step 1 .   
hdfs dfs -mkdir sparksq13   
hdfs dfs -put patients.csv sparksq13/**

**Solution :**

**Step 1 .   
hdfs dfs -mkdir sparksq13   
hdfs dfs -put patients.csv sparksq13/   
Step 2 : Now in spark shell   
//SQLContext entry point tor working with structured data   
val sqlContext = new org.apache.spark.sql.SQLContext(sc)   
//this is used to implicitly convert an RDD to a DataFrame.   
import sqlContext.implicits.   
//Import Spark SQL data types and Row.   
import org.apache.spark.sql.   
//load the data into a new RDD   
val patients = sc.textFile("sparksq13/patients.csv")   
Return the first element in this RDD   
patients.tirst()   
//define the schema using a case class   
case class Patient(patientid: Integer, name: String, dateOfBirth:String , lastVisitDate: String)   
//create an RDD of Product objects   
val patRDD = patients.map(\_.split(",")).map(p => Patient(p(O).tolnt,p(1),p(2),p(3)))   
patRDD.first()   
patRDD.count()**

**//change RDD of Product objects to a DataFrame   
val patDF = patRDD.tODF()**

**//register the DataFrame as a temp table   
patDF.registerTempTable("patients")   
//Select data from table   
val results = sqlContext.sql( """SELECT\* from patients""" )   
//display datatrame in a tabular format   
results.show()   
FROM patients )   
//Find all the patients whose lastVisitDate between current time and   
'2012-09-15'   
val results = sqlContext.sql( """ SELECT\* FROM patients WHERE year(to\_date(cast(unix\_timestamp(lastvisitdate,’yyyy-mm-dd’) as timestamp))between ‘2012-09-15’ and current\_timestamp() order by lastvisitdate”””)  
results.show()   
//Find all the patients who born in 2011   
val results= sqlcontext.sql(“””select\* from patients where year (to\_date(cast(unix\_timesTamp(dateofbirth,’yyyy-mm-dd’) as timestamp))) =2011”””)**

**//find all the patients age   
val results = sqlContext.sql(“””Selectname,dateofbirth,datediff(current\_date(), To\_date(cast(unix\_timestamp(dateofbirth,’yyyy-mm-dd’)as timestamp)))/365 as ageFROM patients   
“””)**

**results. show()   
//List patients whose last visited more than 60 days ago   
--List patients whose last visited more than 60 days ago   
val results = sqcontext.sql( """ SELECT name, lastVßitoate FROM patients WHERE datediff(current\_date(),to\_date(cast(unix\_timestamp(lastvisitdate, ‘yyyy-mm-dd’) as timestamp))).60”””);**

**results. show();   
--Select patients 18 years old or younger   
SELECT\* FROM patients WHERE to\_date(cast(unix\_timestamp(dateofbirth, ‘yyyy-mm-dd’) AS TIMESTAMP)) > date\_sub9current\_date(),INTERval 18year);**

**val results = sqlContext.sql(“””select\* from patients where to\_date(cast(unix\_timestamp(dateofbirth,’yyyy-mm-dd’)as timestamp))>date\_sub(current\_date(),18\*365)”””);   
results. show();   
val results = sqlContext.sql(“””select date\_sub(current\_date(),18\*365)from patients”””);   
results. show();**

1. **Problem Scenario 90 : You have been given below two files   
   course.txt   
   id,course   
   1,Hadoop   
   2, Spark   
   3,HBase   
   Fee.txt   
   id,fee   
   2, 3900   
   3,4200   
   4,2900   
   Accomplish the following activities.   
   1. Select all the courses and their fees , whether tee is listed or not.   
   2. Select all the available fees and respective course. If course does not exists still list the fee   
   3. Select all the courses and their fees , whether tee is listed or not. However, ignore records having tee as null.**

**Solution :**

**Step 1 .   
hdfs dfs -mkdir sparksq14   
hdfs dfs -put course.txt sparksq14/   
hdfs dfs -put tee.txt sparksq14/**

**Step 2 : Now in spark shell   
//load the data into a new ROD   
val course = sc.textFile("sparksq14/course.txt")   
val fee = sc.textFile("sparksq14/fee.txt")   
//Return the first element in this RDD   
course.first()   
fee.first()**

**//define the schema using a case class   
case class Course(id: Integer, name: String)   
case class Fee(id: Integer, fee: Integer)   
//create an RDD ot Product objects   
val courseRDD = course.map(\_.split(",")).map(c =>   
val feeRDD = , => Fee(c(O).tolnt,c(1).tolnt))   
courseRDD.first()   
courseRDD.count()   
feeRDD.first()   
feeRDD.count()   
// change RDD of Product objects to a DataFrame   
val courseDF = courseRDD.toDF()   
val feeDF = feeRDD.toDF()   
// register the DataFrame as a temp table   
courseDF.registerTempTable("course")   
feeDF.registerTempTable("fee")**

**// Select data from table   
val results = sqlContext.sql( """SELECT\* FROM course “"")   
results.show()   
val results = sqlContext.sql( """SELECT\* FROM fee”””)   
results.show()   
val results = sqlContext.sql( """SELECT\* FROM course LEFT JOIN fee ON course.id = fee.id”””)   
results.show()   
val results = sqlContext.sql("""SELECT\* FROM course RIGHT JOIN fee ON course.id = fee.id”””)   
results.show()   
val results = sqlContext.sql("""SELECT\* FROM course LEFT JOIN fee ON course.id = fee.id where fee.id IS NULL”””)   
results.show()**

1. **Problem Scenario 91: You have been given data in json format as below.**

**{"first\_name":"Ankit", “last\_name”. “Jain”}   
{"first\_name":"Amir", “last\_name” . “Khan”}**

**{"first\_name":"Rajesh", "last name":"Khanna"}   
{"first\_name":"Priynka", "last name":"Chopra"}   
{"first\_name":"Kareena", "last name":"Kapoor"}   
{"first\_name":"Lokesh", "last name":"Yadav"}**

**Do the following activity   
1. create employee.json file locally.   
2. Load this tile on hdfs   
3. Register this data as a temp table in Spark using Python.**

**4. Write select query and print this data.   
5. Now save back this selected data in json format.**

**====================================================================**

**Solution**

**Step 1 : create employee.json file locally.   
vi employee.json   
(press insert)   
past the content.**

**Step 2 : Upload this file to hdfs, default location   
hadoop fs -put employee.json**

**val employee = sqlContext.read.json(“/user/cloudera/employee.json”)   
employee.write.parquet("employee.parquet")**

**val parq\_data = sqlcontext.read.parquet("employee.parquet")**

**parq\_data.registerTempTable("employee")**

**val all\_employee = sqlContext.sql("SELeCT\* FROM employee")**

**all\_employee.show()**

**import org.apache.spark.sql.SaveMode**

**prdDF.write.format("orc").saveAsTable("product\_orc\_table")**

**//Change the codec.**

**Option(“compression”,"snappy")**

**employee.write.mode(SaveMode.Overwrite).parquet("employee.parquet")**

1. **Problem Scenario 92 : You have been given a spark scala**

**application, which is bundled in jar named hadoopexam.jar . Your application class name is com.hadoopexam.MyTask**

**You want that while submitting your application should launch a driver on one of the cluster node. Please complete the following command to sut**

**spark-submit \**

**XXX\**

**--master yarn \**

**SPARK\_HOME/lib/hadoopexam.jar 10**

**Solution   
XXX : --class com.hadoopexam.MyTask   
yyy : --deploy-mode cluster**

**--------------------------------------------------------------------**

1. **Problem Scenario 93 : You have to run your Spark application with locally 8 thread or locally on 8 cores. Replace XXX with correct values.**

**spark-submit \**

**--class com.hadoopexam.MyTask \**

**XXX \**

**--deploy-mode cluster SSPARK\_HOME/lib/hadoopexam.jar 10**

**Solution**

**XXX : --master local[8]**

**Notes : The master URL passed to Spark can be in one ot the following formats:   
Master URL Meaning   
local:**

**Run Spark locally with one worker thread (i.e. no parallelism at all).   
local[K]:**

**Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).   
localn [\*]:**

**Run Sparkocally with as many worker threads as logical cores on your machine.   
spark://HOST:PORT:**

**Connect to the given Spark standalone cluster master. The port must be whichever one your master is configured to use, which is 7077 by default.   
mesos://HOST:PORT:**

**Connect to the given Mesos cluster. The port must be whichever one your is contigured to use, which is 5050 by default. Or, for a Mesos   
cluster using ZooKeeper, use mesos://zk://.... To submit with --deploy-mode cluster, the HOST:PORT should be configured to connect to the MesosClusterDispatcher.   
yarn:**

**Connect to a YARN cluster in client or cluster mode depending on the value ot --deploy-mode. The cluster location will be found based on the HADOOP\_CONF\_DIR or YARN\_CONF\_DIR variable.**

**====================================================================**

1. **Problem Scenario 94 : You have to run your Spark application on yarn with each executor 20GB and number ot executors should be 50. Please replace XXX, YYY, ZZZ**

**export HADOOP\_CONF\_DIR=$HADOOP\_HOME/conf**

**spark-submit \   
--class com.hadoopexam.MyTask \   
xxx \**

**yyy \   
--deploy-mode cluster \  
zzz \   
/path/to/hadoopexam.jar 1000**

**Solution**

**XXX : --master yarn   
YYY : --executor-memory 20G   
ZZZ : --num-executors 50**

**====================================================================**

1. **Problem Scenario 95 : You have to run your Spark application on yarn with each executor Maximum heap size to be 512MB and Number ot processor cores to allocate on each executor will be 1 and your main application required three values as input arguments V1,V2,V3.pleasee replace XXX, YYY, ZZZ**

**spark-submit \**

**--class com.hadoopexam.MyTask \**

**--master yarn-cluster \**

**--num-executors 3 \**

**--driver-memory 512m \**

**XXX \**

**YYY \**

**lib/hadoopexam.jar ZZZ**

**Solution   
XXX : --executor-memory 512m   
YYY : --executor-cores 1   
zzz : V1 V2 V3   
Notes : spark-submit on YARN Options   
Option Description   
archives:**

**Comma-separated list ot archives to be extracted into the working directory oF each executor. The path must be globally visible inside Your cluster; see Advanced Dependency Management.  
executor-cores:**

**Number ot processor cores to allocate on each executor. Alternatively, you can use the spark.executor.cores property.   
executor-memory:**

**Maximum heap size to allocate to each executor. Alternatively, you can use the spark.executor.memory property.   
num-executors:**

**Total number ot YARN containers to allocate for this application. Alternatively, you can use the spark.executor.instances property. property.   
queue:**

**YARN queue to submit to. For more information, see Assigning Applications and Queries to Resource Pools.   
Detault: default.**

**====================================================================**

1. **Problem Scenario 96 :**

**Your spark application required extra Java options as below.   
-xx:+printGCDetails -xx:+printGcTimestamps   
Please replace the XXX values correctly**

**spark-submit --name "My app" --master local[4] \**

**--conf spark.eventLog.enabled=false \  
--conf XXX \**

**hadoopexam.jar**

**Solution:**

**XXX :**

**"spark.executor.extraJavaOptions=**

**-XX:+PrintGCDetails -XX:+PrintGCTimeStamps”  
Notes :   
spark-submit \**

**--class <main-class>   
--master <master-url> \   
--deploy-mode <deploy-mode> \   
--conf <key>=<value> \   
# other options   
<application-jar> \   
[application-arguments]   
Here, conf is used to pass the Spark related configs which are required for the application to run like any specific property(executor memory) or if you want to override the default property which is set in spark-default.conf.**