Starting JUMBO application

1. Open Ubuntu Terminal and Type the following commands to start the Hadoop server

    1.a. cd /usr/local/hadoop/sbin

    1.b. start-all.sh

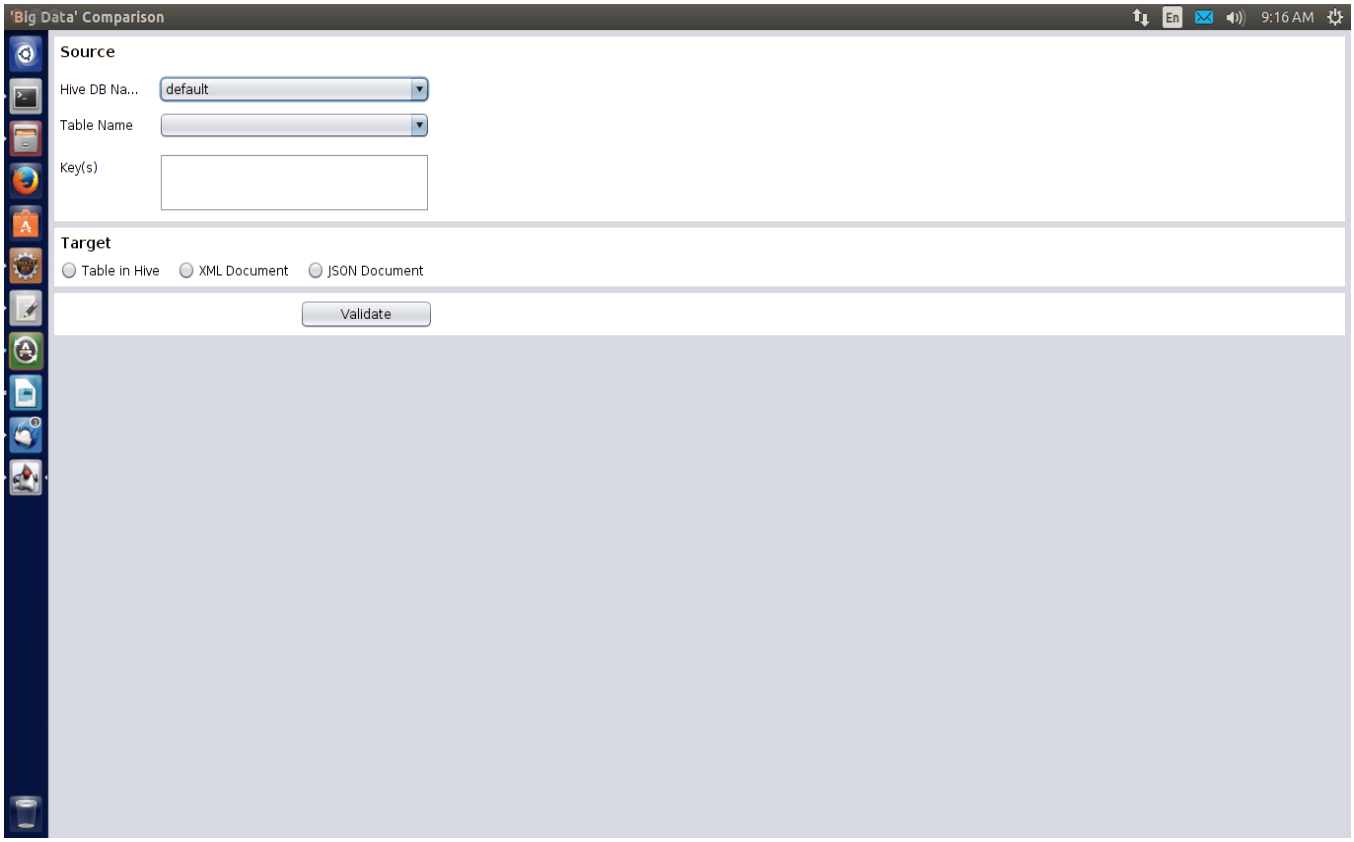
    1.c. mr-jobhistory-daemon.sh start historyserver

    1.d. hive --service metastore

2. Start the application using Eclipse

Overview of the tool

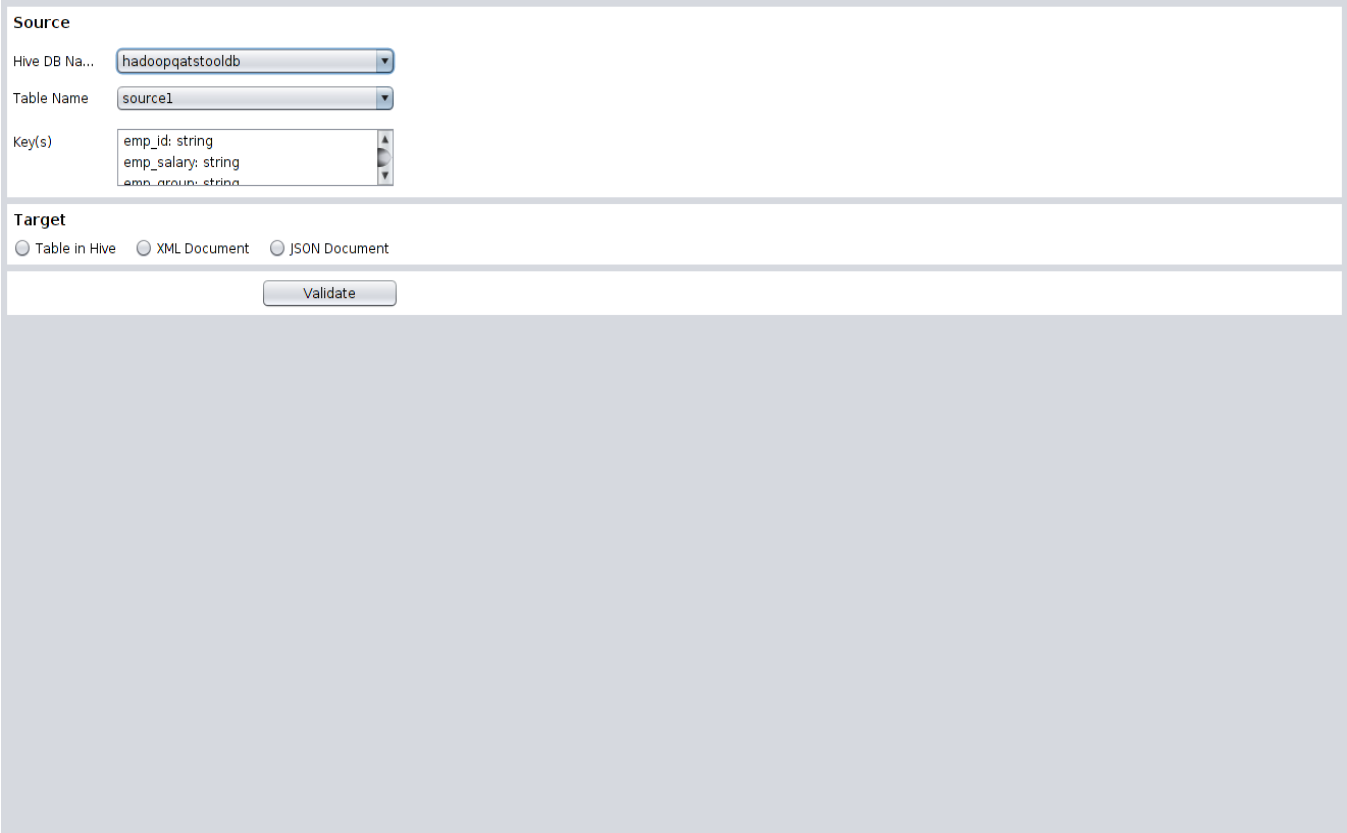
1. UI



The initial look and feel of the UI is shown above. This panel consists of a Source Database related panel and Target Source Panel.

Under Source panel, The Hive DB Name is a Dropdown list which contains all the database names present in the hive database.

All the tables, related to the selected database will be populated in the table name dropdown box and the columns of the selected tables are populated in the key(s) list

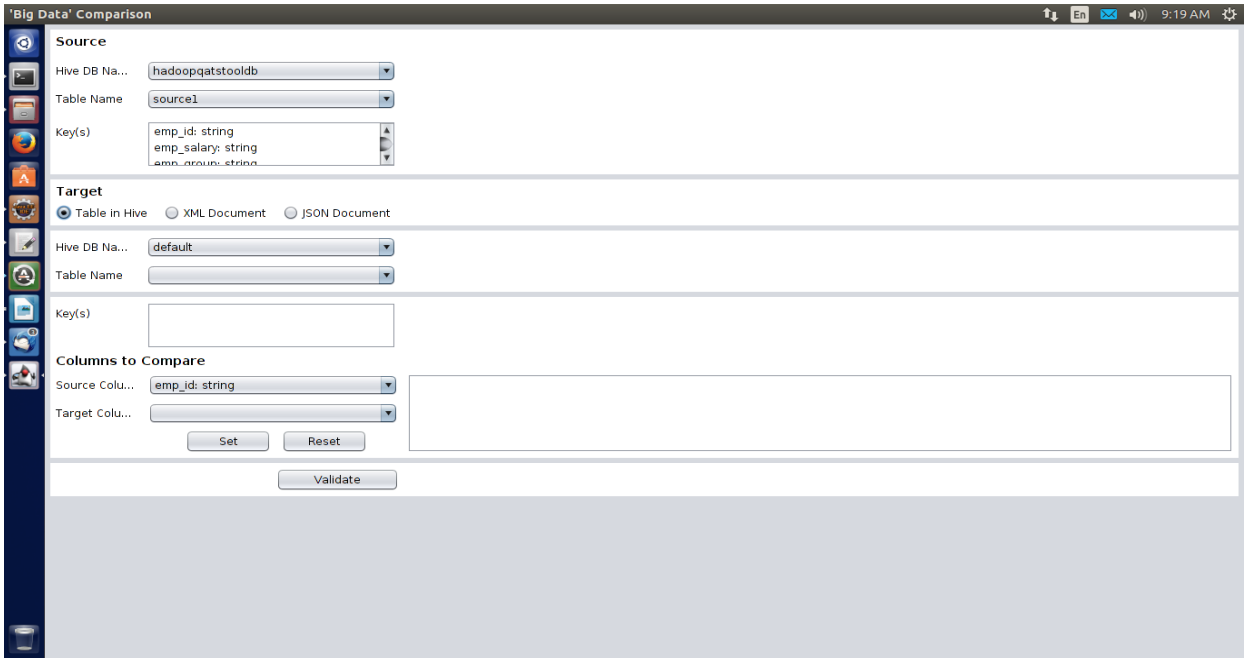


dynamically.

In the Target Source Panel, there are 3 radio buttons, namely Table in Hive, XML Document, JSON Document.

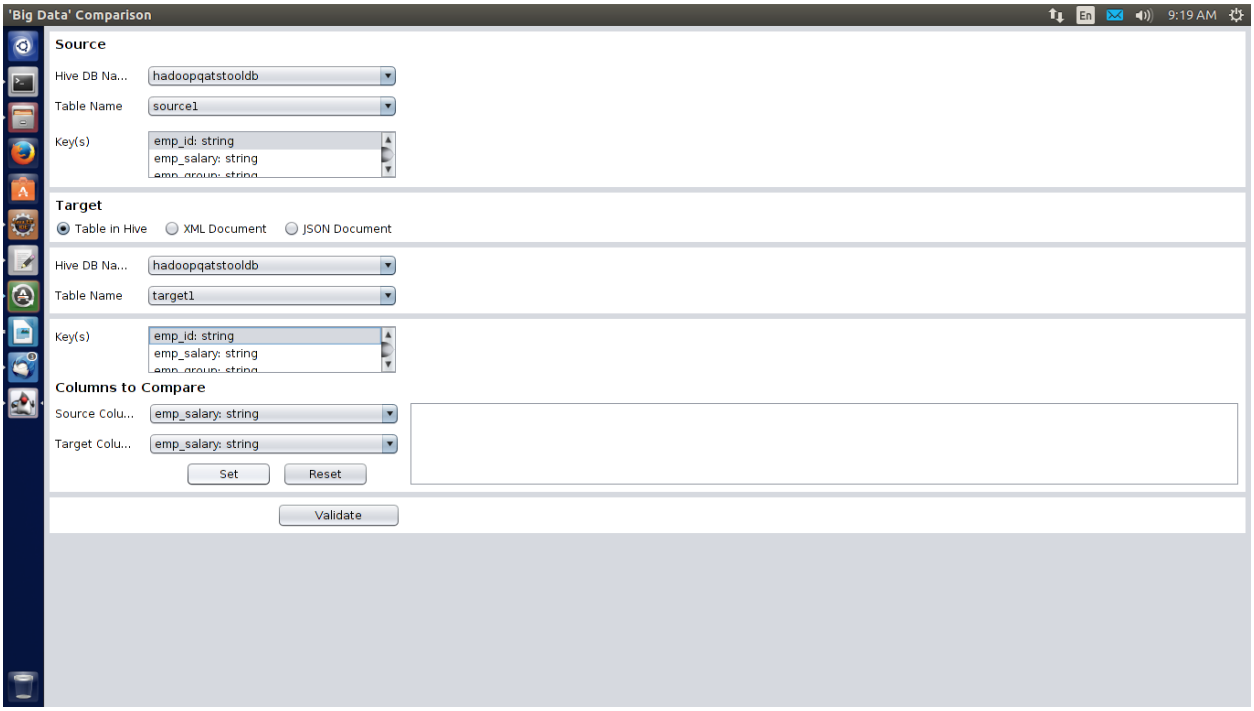
On selection of individual radio buttons, different panels are generated.

2. Target as Table in Hive



After selecting the Table in Hive radio button, we get a window as displayed below.

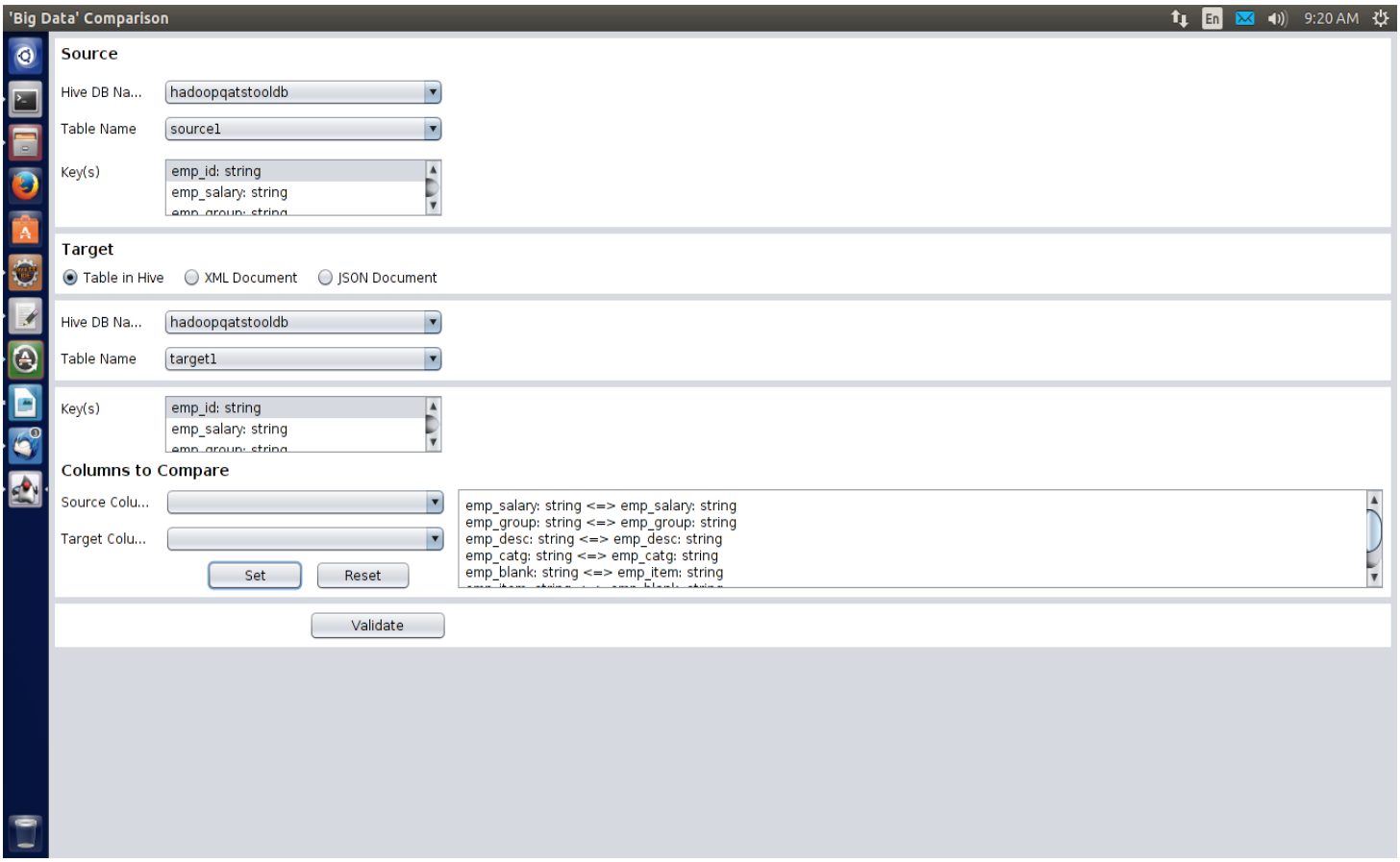
Here we have the same panels as earlier, additionally we have a Target Panel and



Columns to Compare panel.Target Panel contains the same features as that of the Source panel.

Now we have to select Keys from source and target key(s). It will create the relationship between the two tables.

In the Columns to compare panel, we have Source Column and Target Column dropdown boxes, which contains the column details respectively.
We have a Set button and a Reset Button.
On clicking the Set button, the selected column names in the two dropdown lists will be taken for comparison. Further that parameters will be displayed in the text field in the current panel.

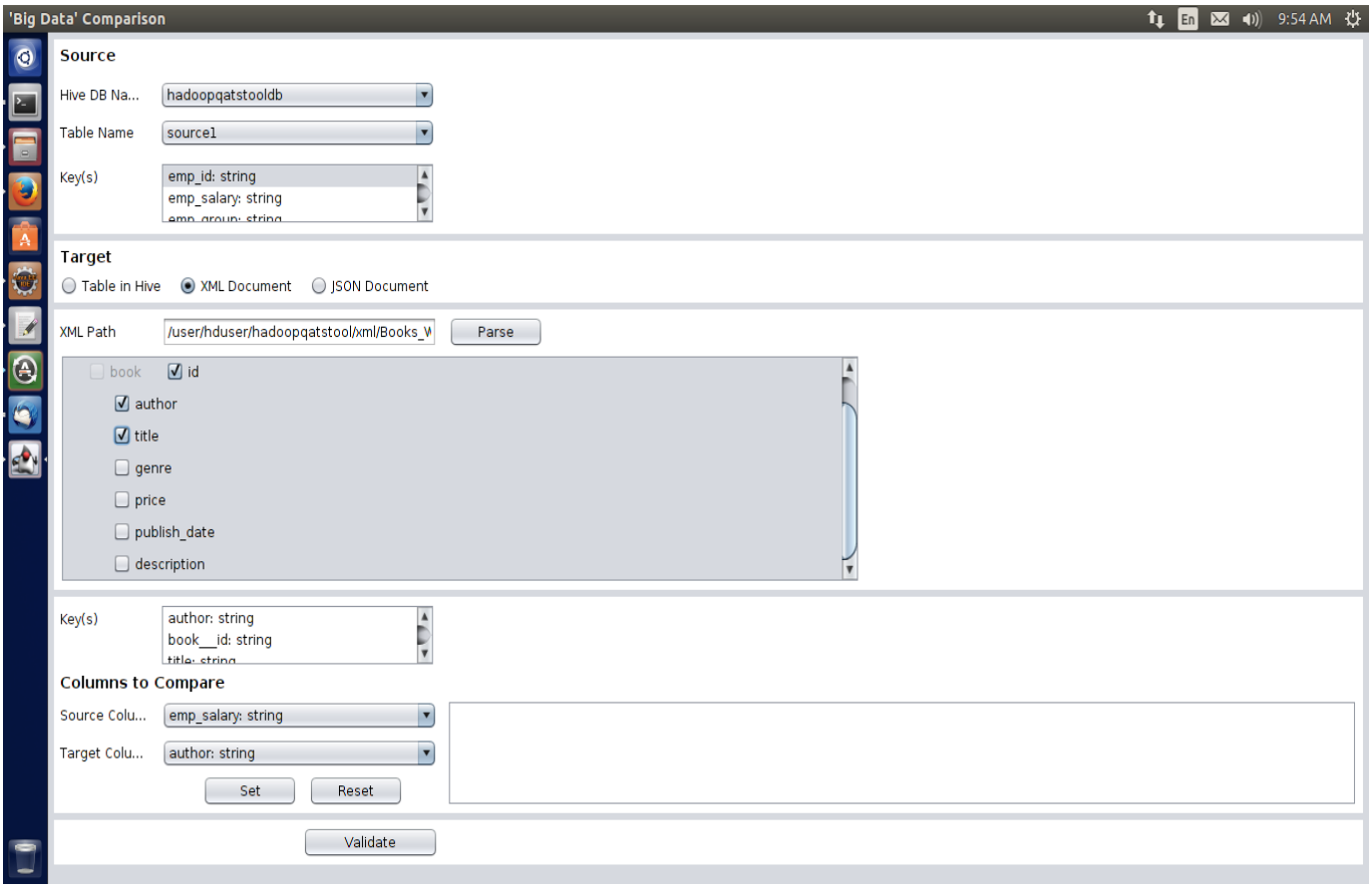


After clicking on set button, the selected parameters move out from the dropdown list and the remaining column names will be displayed in the respective dropdown lists.

On clicking Reset Button, All the selected parameters in the text field will be removed and loaded back to their respective dropdown boxes.

3. Target as XML Document

After selecting the XML document radio button, we get a window as displayed



below.

It consists of an XML File Parsing panel and Columns To Compare column.

XML File parsing column consists of a field called XML Path. Here the XML File path should be a
hdfs file path and it should be an XML file only.

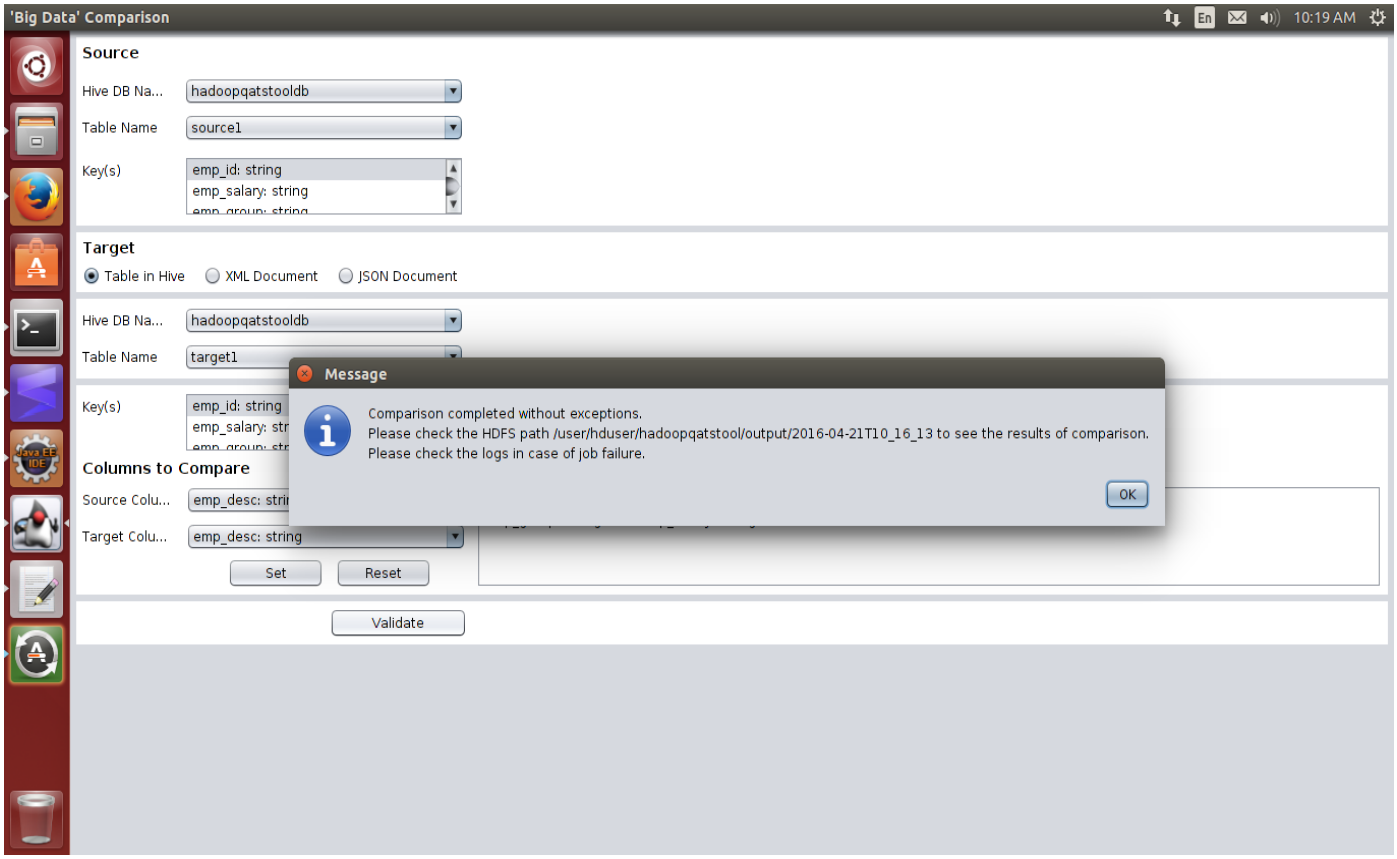
Clicking on Parse button will Parse the selected XML File i.e retrieves all the node details i.e the column names and displays inside the panel below with a check box in front of it.

In the key(s) list, only those column names and details will be displayed which has been selected in the above panel.

4. **Validation**
   Click on Validate to start validation.

5. **Results**

Result panel will be generated as soon as the validation is complete followed by a

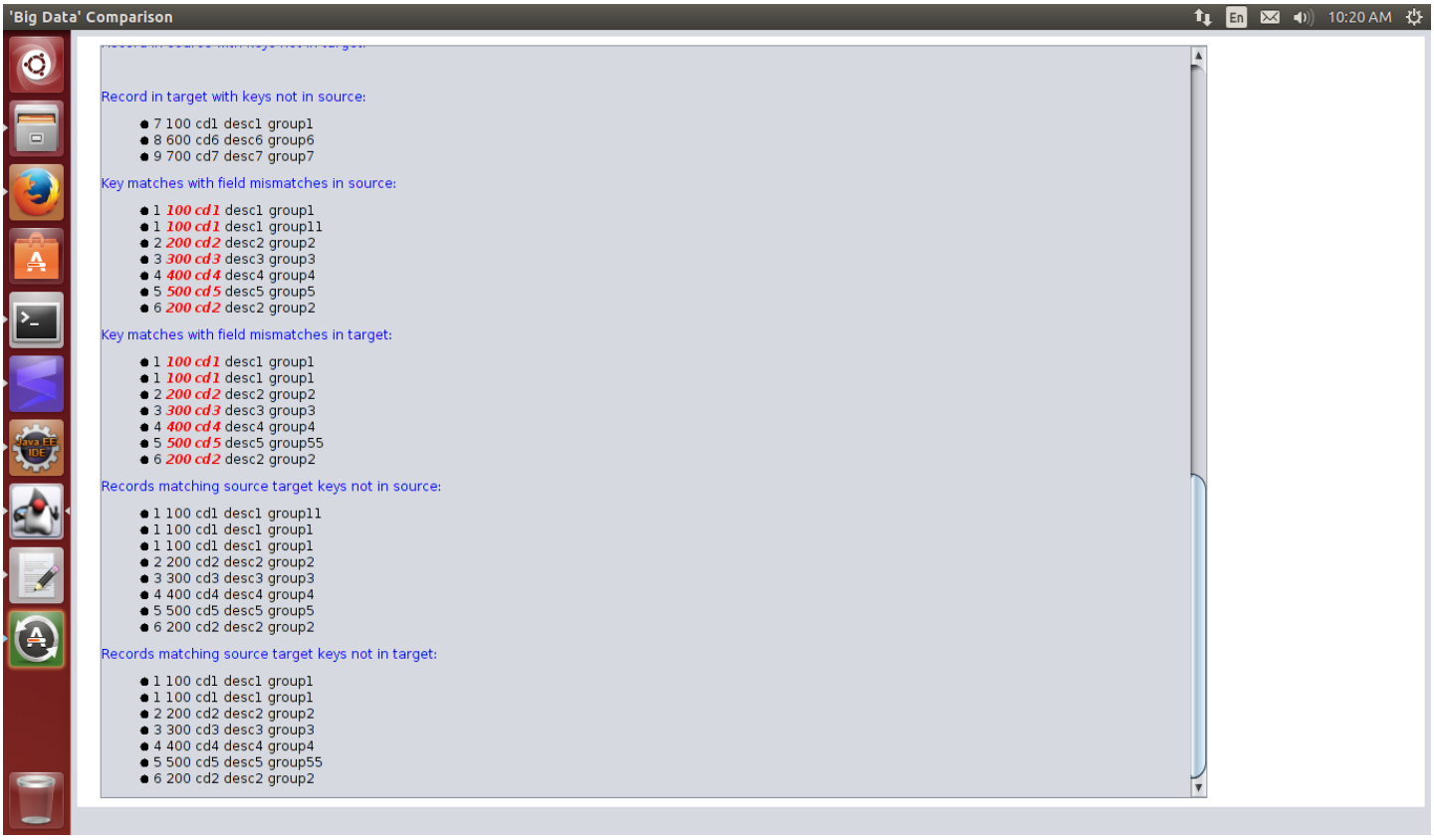


Pop up message saying Comparison Completed without Exceptions.

Clicking on Ok will display the Result log as shown below.


'Big Data' Comparison
10:19 AM
hdfs output path:
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/duplicate_records_in_source_by_keys
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/duplicate_records_in_source_entire_row
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/duplicate_records_in_target_by_keys
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/duplicate_records_in_target_entire_row
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/key_matches_with_field_mismatches_in_source
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/key_matches_with_field_mismatches_in_target
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/records_in_source_with_keys_not_in_target
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/records_in_target_with_keys_not_in_source
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/records_matching_src_tgt_keys_in_source
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/records_matching_src_tgt_keys_in_target
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/statistics
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/unique_records_in_source_by_keys
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/unique_records_in_source_entire_row
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/unique_records_in_target_by_keys
hdfs://localhost:54310/user/hduser/hadoopqatstool/output/2016-04-21T10_16_13/unique_records_in_target_entire_row
Statistics:
Total Number of records in target with Key(s) missing in source: 3
Total Number of Duplicate Records in Source (by entire row): 2
Total Number of Duplicate Records in Target (by entire row): 2
Total Number of Unique Records in Source (by entire row): 6
Total Number of Unique Records in Target (by entire row): 8
Total Number of Duplicate Records in Source (by key(s)): 3
Total Number of Duplicate Records in Target (by key(s)): 2
Total Number of records in source with matching key(s): 8
Total Number of records in target with matching key(s): 7
Total Number of Unique Records in Source (by key(s)): 5
Total Number of Unique Records in Target (by key(s)): 8
Total Number of Key(s) matching in Source and Target: 6
Total Number of Key(s) in target missing in source: 3
Total Number of Records in Target: 10
Total Number of Records in Source: 8
Duplicate records in source by keys:
1 100 cd1 desc1 group1
null
Duplicate records in target by keys:
1 100 cd1 desc1 group1
1 100 cd1 desc1 group1
Unique records in source by keys:
2 200 cd2 desc2 group2

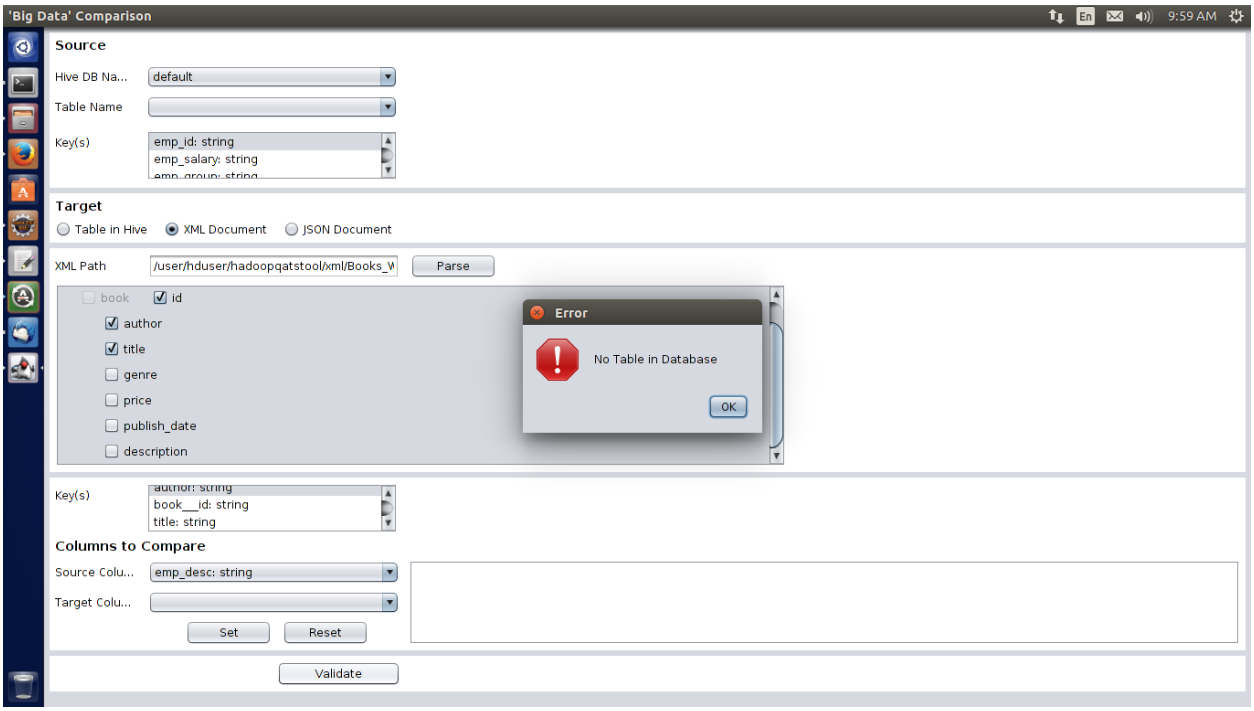

'Big Data' Comparison
10:20 AM
Duplicate records in target by keys:
1 100 cd1 desc1 group1
1 100 cd1 desc1 group1
Unique records in source by keys:
2 200 cd2 desc2 group2
3 300 cd3 desc3 group3
4 400 cd4 desc4 group4
5 500 cd5 desc5 group5
6 200 cd2 desc2 group2
Unique records in target by keys:
2 200 cd2 desc2 group2
3 300 cd3 desc3 group3
4 400 cd4 desc4 group4
5 500 cd5 desc5 group55
6 200 cd2 desc2 group2
7 100 cd1 desc1 group1
8 600 cd6 desc6 group6
9 700 cd7 desc7 group7
Record in source with keys not in target:
Record in target with keys not in source:
7 100 cd1 desc1 group1
8 600 cd6 desc6 group6
9 700 cd7 desc7 group7
Key matches with field mismatches in source:
1 100 cd1 desc1 group1
1 100 cd1 desc1 group11
2 200 cd2 desc2 group2
3 300 cd3 desc3 group3
4 400 cd4 desc4 group4
5 500 cd5 desc5 group5
6 200 cd2 desc2 group2
Key matches with field mismatches in target:
1 100 cd1 desc1 group1
1 100 cd1 desc1 group1
2 200 cd2 desc2 group2
3 300 cd3 desc3 group3



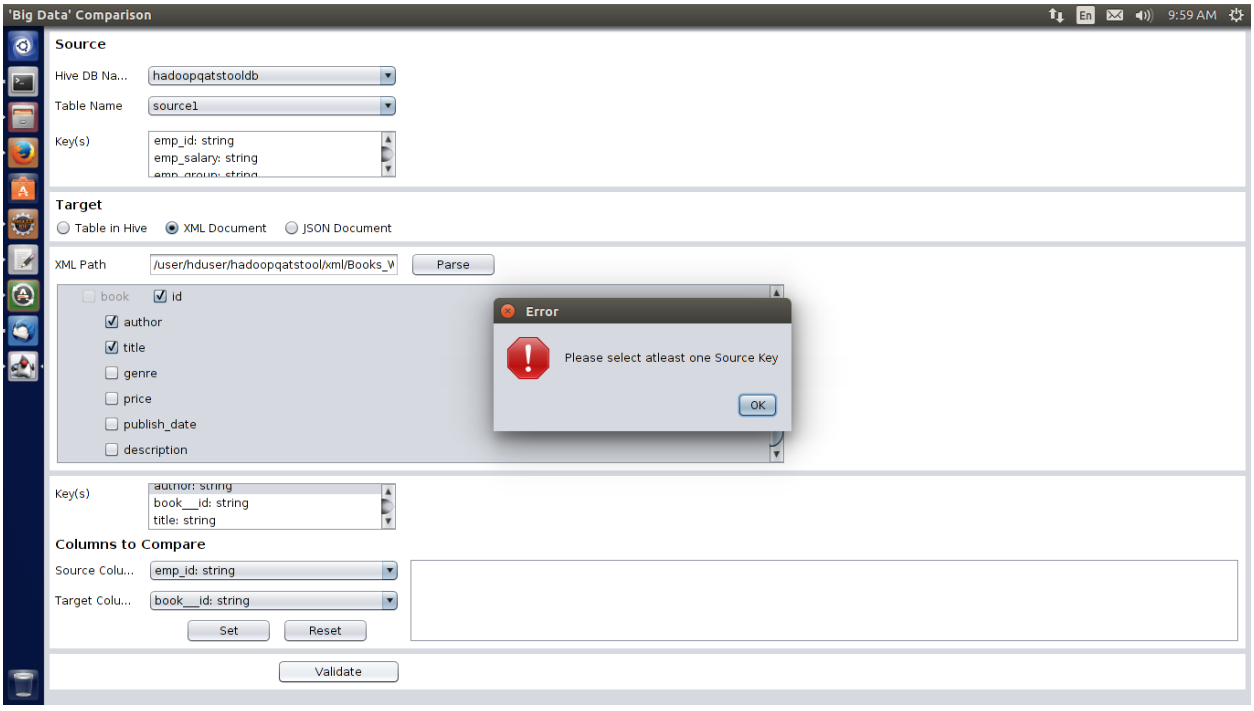
Mismatch records can be seen in **BOLD RED** font in the UI.

Possible Error Scenarios

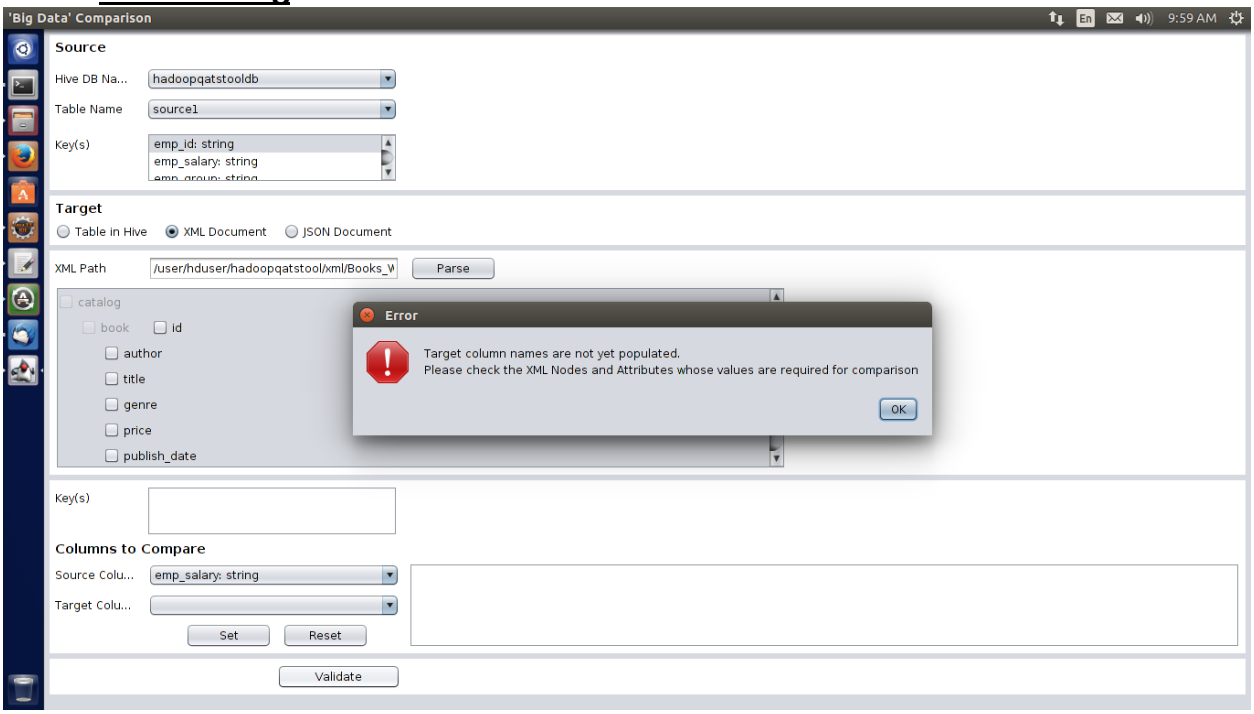
1. If there is no table in the selected database



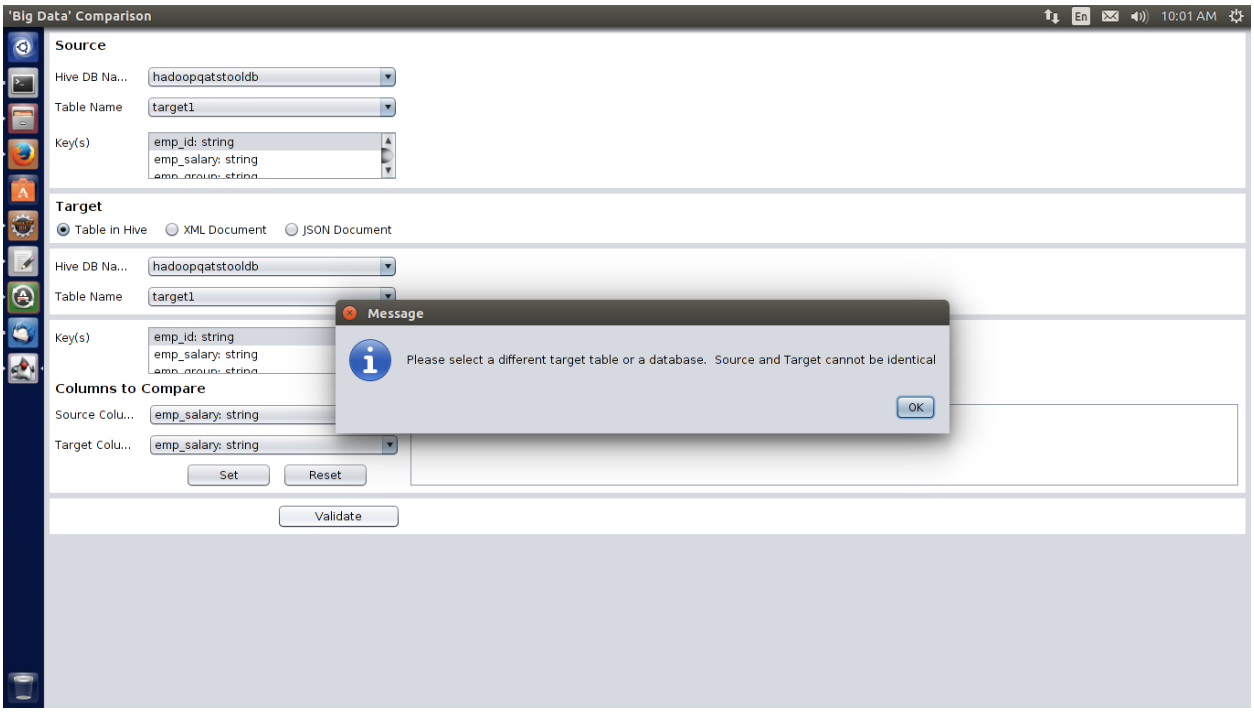
2. If source keys in the source and target key(s) has not been selected.



3. **If none of the columns has been checked, that has been populated after XML Parsing**



4. **If source and target tables as well as databases are same**



HADOOP SET UP

Step 1: Installing Java

Hadoop framework is written in Java

Update the source list

```
user@system:~$ sudo apt-get update
```

The OpenJDK project is the default version of Java

that is provided from a supported Ubuntu repository.

```
user@system:~$ sudo apt-get install default-jdk
user@system:~$ java -version
java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.3) (7u71-2.5.3-0ubuntu0.14.04.1)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Step 2: Adding a dedicated Hadoop user

```
user@system:~$ sudo addgroup hadoop
Adding group `hadoop' (GID 1002) ...
Done.
user@system:~$ sudo adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

```
Changing the user information for hduser
Enter the new value, or press ENTER for the default
      Full Name []:
      Room Number []:
      Work Phone []:
      Home Phone []:
      Other []:
Is the information correct? [Y/n] Y
```

Step 3: Installing SSH

ssh has two main components:

ssh: The command we use to connect to remote machines - the client.

sshd: The daemon that is running on the server and allows clients to connect to the server.

The ssh is pre-enabled on Linux, but in order to start sshd daemon, we need to install sshfirst. Use this command to do that :

```
user@system:~$ sudo apt-get install ssh
```

Create and Setup SSH Certificates

Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

```
user@system:~$ su hduser
Password:
hduser@laptop:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
```

```
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
50:6b:f3:fc:0f:32:bf:30:79:c2:41:71:26:cc:7d:e3 hduser@laptop
The key's randomart image is:
+--[ RSA 2048]----+
|        .oo.o    |
|       . .o=. o  |
|      . + .  o . |
|       o =    E  |
|        S +      |
|         . +     |
|          O +    |
|           O o   |
|            o..  |
+-----------------+
```

su- this command is use to switch user(i.e from normal user to hduser)

The following command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.

```
hduser@laptop:/home/k$ cat $HOME/.ssh/id_rsa.pub >>
$HOME/.ssh/authorized_keys
```

We can check if ssh works:

```
hduser@laptop:/home/k$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is
e1:8b:a0:a5:75:ef:f4:b4:5e:a9:ed:be:64:be:5c:2f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known
hosts.
```

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-40-generic x86_64)
...
```

Step 4: Install Hadoop

```
hduser@laptop:~$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
hduser@laptop:~$ tar xvzf hadoop-2.6.0.tar.gz
```

xvzf- this command will extract the tar file.

We want to move the Hadoop installation to the /usr/local/hadoop directory , normally we cannot able to move so we are giving the root priviledge to hduser using the following command:

```
hduser@laptop:~/hadoop-2.6.0$ su k
Password:
user@system:/home/hduser$ sudo adduser hduser sudo
[sudo] password for k:
Adding user `hduser' to group `sudo' ...
Adding user hduser to group sudo
```

Done.

Now we can move the Hadoop installation to the /usr/local/hadoop directory without any problem:

```
user@system:/home/hduser$ sudo su hduser
hduser@laptop:~/hadoop-2.6.0$ sudo mv * /usr/local/hadoop
hduser@laptop:~/hadoop-2.6.0$ sudo chown -R hduser:hadoop /usr/local/hadoop
```

The hadooop installation part is over there are certain configuration we need to done before running hadoop.

Setup Configuration Files

1. ~/.bashrc:

```
hduser@laptop:~$ vi ~/.bashrc
```

vi – using this command we can edit the files in terminal. The following commands are to be added to the bashrc file.

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

```
hduser@laptop:~$ source ~/.bashrc
```

source – using this command we can save the edited files.

2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Adding the above statement in the hadoop-env.sh file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

3. /usr/local/hadoop/etc/hadoop/core-site.xml:

The /usr/local/hadoop/etc/hadoop/core-site.xml file contains configuration properties that Hadoop uses when starting up.This file can be used to override the default settings that Hadoop starts with.

```
hduser@laptop:~$ sudo mkdir -p /app/hadoop/tmp
hduser@laptop:~$ sudo chown hduser:hadoop /app/hadoop/tmp
```

Open the file and enter the following in between the <configuration></configuration> tag:

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/core-site.xml

<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>

 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
 </property>
</configuration>
```

4. /usr/local/hadoop/etc/hadoop/mapred-site.xml

By default, the /usr/local/hadoop/etc/hadoop/ folder contains /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
file which has to be renamed/copied with the name mapred-site.xml:

```
hduser@laptop:~$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

The mapred-site.xml file is used to specify which framework is being used for MapReduce.
We need to enter the following content in between the <configuration></configuration> tag:

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
 </property>
</configuration>
```

5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

The /usr/local/hadoop/etc/hadoop/hdfs-site.xml file needs to be configured for each host in the cluster that is being used.It is used to specify the directories which will be used as the namenodeand thedatanodeon that host.Before editing this file, we need to create two directories which will contain the namenode and the datanode for this Hadoop installation.
This can be done using the following commands:

```
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
hduser@laptop:~$ sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

Open the file and enter the following content in between the <configuration></configuration> tag:

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml


<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
<description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
 </property>
 <property3. /usr/local/hadoop/etc/hadoop/core-site.xml:


The     /usr/local/hadoop/etc/hadoop/core-site.xml     file     contains configuration properties that Hadoop uses when starting up.
This file can be used to override the default settings that Hadoop starts with.


hduser@laptop:~$ sudo mkdir -p /app/hadoop/tmp
hduser@laptop:~$ sudo chown hduser:hadoop /app/hadoop/tmp
Open the file and enter the following in between the
<configuration></configuration> tag:


hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/core-site.xml


<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
```

```
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
 </property>

 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
determine the host, port, etc. for a filesystem.</description>
 </property>
</configuration>
```

4. /usr/local/hadoop/etc/hadoop/mapred-site.xml

By default, the /usr/local/hadoop/etc/hadoop/ folder contains/usr/local/hadoop/etc/hadoop/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml:

```
hduser@laptop:~$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

```
The mapred-site.xml file is used to specify which framework is being used for MapReduce.
```

```
We need to enter the following content in between the <configuration></configuration> tag:
```

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
```

```
  and reduce task.
  </description>
 </property>
</configuration>
```

5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

The /usr/local/hadoop/etc/hadoop/hdfs-site.xml file needs to be configured for each host in the cluster that is being used.

It is used to specify the directories which will be used as the namenode and the datanode on that host.

Before editing this file, we need to create two directories which will contain the namenode and the datanode for this Hadoop installation.

This can be done using the following commands:

```
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
hduser@laptop:~$ sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

Open the file and enter the following content in between the <configuration></configuration> tag:

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

```
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
```

```
  </description>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
 </property>
</configuration>>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
 </property>
</configuration>
```

Step 5: Format the New Hadoop Filesystem

Now, the Hadoop file system needs to be formatted so that we can start to use it. The format command should be issued with write permission since it creates current directory
under /usr/local/hadoop_store/hdfs/namenode folder:

The format command is as follows,

```
hduser@laptop:~$ hadoop namenode -format
```

Step 6: Pig installation

Download the last Pig release http://pig.apache.org/releases.htmlExact mirror for me was http://apache-mirror.rbc.ru/pub/apache/pig/pig-0.13.0/pig-0.13.0.tar.gz

Enter into the directory where the stable version is downloaded. By default it downloads in “Downloads” directory.

```
$ cd Downloads/
```

Unzip the tar file.

```
$ tar -xvf pig-0.15.0.tar.gz
```

Create directory using mkdir command

```
$ sudo mkdir /usr/lib/pig
```

move pig-0.11.1 to pig using mv command

```
$ mv pig-0.15.0 /usr/lib/pig/
```

Set the PIG_HOME path in bashrc file

```
$ vi ~/.bashrc
```

In bashrc file append the below 2 statements

```
export PIG_HOME=/usr/lib/pig/pig-0.15.0
export PATH=$PATH:$PIG_HOME/bin
```

Now Pig installation part is done.

Step 7: Hive Installation

Browse to the link: http://apache.claz.org/hive/stable/

Click the apache-hive-1.2.1-bin.tar.gz download it

```
$  cd  /usr/lib/

$  sudo mkdir hive

$  cd Downloads

$  sudo mv apache-hive-1.2.1-bin /usr/lib/hive

$ vi ~/.bashrc
```

now Copy and paste the following lines at end of the bashrc file

```
# Set HIVE_HOME

export HIVE_HOME="/usr/lib/hive/apache-hive-0.13.0-bin"

PATH=$PATH:$HIVE_HOME/bin

export PATH
```

Setting HADOOP_PATH in HIVE config.sh

```
$ cd  /usr/lib/hive/apache-hive-1.2.1-bin/bin

$ vi hive-config.sh
```

now write the path where hadoop file is there

export HADOOP_HOME=/usr/local/hadoop

Create Hive directories within HDFS

```
$   hadoop fs -mkdir /usr/hive/warehouse
```

Setting READ/WRITE permission for table

```
$  hadoop fs -chmod g+w /usr/hive/warehouse
```

Step 8: Configuring the Hive Metastore Database

Install and start MySQL if you have not already done so

1. To install MySQL on a Debian/Ubuntu system:
   ```
   $sudo apt-get install mysql-server
   ```
2. After using the command to install MySQL, you may need to respond to prompts to confirm that you do want to complete the installation. After installation completes, start the mysql daemon.
   ```
   $sudo service mysql start
   ```
3. Configure the MySQL service and connector

   To install the MySQL connector on a Debian/Ubuntu system:
   ```
   $sudo apt-get install libmysql-java

   $cp /usr/share/java/mysql.jar /usr/local/hive/lib

   $cp                /usr/share/java/mysql-connector-java-5.1.28.jar
   /usr/local/hive/lib
   ```
4. Create the database and user
   ```
   $ mysql -u root -p

   Enter password:

   mysql> CREATE DATABASE metastore;
   ```

```
mysql> USE metastore;

mysql> SOURCE /usr/lib/hive/scripts/metastore/upgrade/mysql/hive-schema-1.2.0.mysql.sql;

mysql> CREATE USER 'hive'@'localhost' IDENTIFIED BY 'root';

...

mysql> REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'hive'@'localhost';

mysql> GRANT ALL PRIVILEGES ON metastore.* TO 'hive'@'localhost';

mysql> FLUSH PRIVILEGES;

mysql> quit;
```

5. Configure the metastore service to communicate with the MySQL database

```
$cd  /usr/local/hive/conf

$touch  hive-site.xml

$vi hive-site.xml

<configuration>

<property>

  <name>javax.jdo.option.ConnectionURL</name>

  <value>jdbc:mysql://localhost/metastore</value>

  <description>the URL of the MySQL database</description>

</property>


<property>

  <name>javax.jdo.option.ConnectionDriverName</name>

  <value>com.mysql.jdbc.Driver</value>

</property>


<property>

  <name>javax.jdo.option.ConnectionUserName</name>

  <value>hive</value>

</property>
```

```
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>root</value>
  </property>

  <property>
 <name>datanucleus.autoCreateSchema</name>
    <value>false</value>
  </property>

  <property>
    <name>datanucleus.fixedDatastore</name>
    <value>true</value>
  </property>

  <property>
    <name>datanucleus.autoStartMechanism</name>
    <value>SchemaTable</value>
  </property>

  <property>
    <name>hive.metastore.uris</name>
    <value>thrift://localhost:9083</value>
    <description>IP address (or fully-qualified domain name) and port
  of the metastore host</description>
</property>
  </configuration>

  <property>
  <name>hive.metastore.schema.verification</name>
```

```
<value>true</value>
</property>
```

Step 9: Loading file from local to hadoop file system

STEP 1: CREATE A DIRECTORY IN HDFS

hadoop fs -mkdir – use to create as directory

hadoop fs -mkdir <path/newfolder_name>

EX: **hadoop fs -mkdir /user/hduser/hadoopqatstool/input**

STEP 2 : UPLOAD FILES FROM LOCAL PATH TO HDFS

hadoop fs -put – used to copy files from local to hdfs

hadoop fs -put /home/Desktop/sorce/user/hduser/hadoopqatstool/input

For XML File Loading, create a new directory as xml.

hadoop fs -put /home/hduser/Desktop/Books_WS_MM.xml /user/hduser/hadoopqatstool/xml

(To copy files)

hadoop fs -ls <hdfs_path> To list out all the files in the path

PIG Scripts for Reference

The tables are compared using pig in the hadoop environment the following pig scripts are used based on the recuirement of table comparison.

-- intially the source and target tables are need to loaded in to the pig

1. source = load '/user/himanshu/hadoopqatstool/input/source1' using PigStorage(',') as(segment_id:chararray,engagement_id:chararray,segment_cd:chararray,segment_desc:chararray,ey_segment_group:chararray);

1-1. dump source;
-- dump command is used to display the output
OUTPUT:
(1,100,cd1,desc1,group1)
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group5)
(6,200,cd2,desc2,group2)
(1,100,cd1,desc1,group1)

2. target = load '/user/himanshu/hadoopqatstool/input/target1' using PigStorage(',')
as(segment_id:chararray,engagement_id:chararray,segment_cd:chararray,segment_desc:chararra
y,ey_segment_group:chararray);
2-1. dump target;
OUTPUT:
(1,100,cd1,desc1,group1)
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group55)
(6,200,cd2,desc2,group2)
(1,100,cd1,desc1,group1)
(7,100,cd1,desc1,group1)

-- To find the total number of records in source
-- total_records_src1 = group source all;
-- describe total_records_src1;
-- OUTPUT:total_records_src1: {group: chararray,source: {(segmellnt_id:
chararray,engagement_id: chararray,segment_cd: chararray,segment_desc:
chararray,ey_segment_group: chararray)}}
-- illustrate total_records_src1;

-- OUTPUT:

| source | segment_id:chararray | engagement_id:chararray | segment_cd:chararray |
segment_desc:chararray | ey_segment_group:chararray |

| | 6 | 200 | cd2 | desc2 | group2
|
| | 2 | 200 | cd2 | desc2 | group2
|

```
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
| total_records_src1     | group:chararray    |
source:bag{:tuple(segment_id:chararray,engagement_id:chararray,segment_cd:chararray,segment
_desc:chararray,ey_segment_group:chararray)}                    |
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
|                    | all                | {(6, ..., group2), (2, ..., group2)}
|
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
```

```
-- dump total_records_src1;
--
OUTPUT:(all,{(1,100,cd1,desc1,group1),(6,200,cd2,desc2,group2),(5,500,cd5,desc5,group5),(4,
400,cd4,desc4,group4),(3,300,cd3,desc3,group3),(2,200,cd2,desc2,group2),(1,100,cd1,desc1,gro
up1)})

--  count the number of records in source
-- Inorder to find the total recordsin a table we need to group the source and then count using
COUNT
3. total_records_src = foreach (group source all) generate CONCAT('Total Number of Records in
Source: ',' ',(chararray)COUNT(source)) as no_of_source_records;
-- CONCAT command will join all things with in it.
3-1. dump total_records_src;
OUTPUT:
(Total Number of Records in Source:  7)

--  count the number of records in target
4. total_records_tgt = foreach (group target all) generate CONCAT('Total Number of Records in
Target: ',' ',(chararray)COUNT(target)) as no_of_target_records;
4-1. dump total_records_tgt;
OUTPUT:
(Total Number of Records in Target:  8)

-- Split duplicate and unique records by entire record in source
-- grpd_src_by_all_columns = group source by ($INPUT_SOURCE_FIELDS);
5. grpd_src_all_columns_recordset_with_count = foreach (group source by
(segment_id,engagement_id,segment_cd,segment_desc,ey_segment_group)) generate source,
COUNT(source) as no_of_records;
5-1. dump grpd_src_all_columns_recordset_with_count;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)
({(2,200,cd2,desc2,group2)},1)
```

({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group5)},1)
({(6,200,cd2,desc2,group2)},1)

6. split grpd_src_all_columns_recordset_with_count into source_unique if no_of_records == 1, source_duplicate if no_of_records > 1, source_no_records if no_of_records == 0;
6-1. dump source_unique;
OUTPUT:
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group5)},1)
({(6,200,cd2,desc2,group2)},1)

6-2. dump source_duplicate;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)

6-3. dump source_no_records;
OUTPUT:

-- extracts the unique records in source

7. source_unique_records = foreach source_unique generate FLATTEN(source);

7-1. dump source_unique_records;
OUTPUT:
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group5)
(6,200,cd2,desc2,group2)

--extracts the duplicate records in source

8. source_duplicate_records = foreach source_duplicate generate FLATTEN(source);

8-1. dump source_duplicate_records;
OUTPUT:
(1,100,cd1,desc1,group1)
(1,100,cd1,desc1,group1)

-- Total number of unique records by entire record in source
-- total_source_unique1 = group source_unique all;

9. total_source_unique = foreach (group source_unique all) generate CONCAT('Total Number of Unique Records in Source (by entire row): ',' ',(chararray)SUM(source_unique.no_of_records)) as no_of_unique_records_in_source_by_row;

9-1. dump total_source_unique;
OUTPUT:
(Total Number of Unique Records in Source (by entire row): 5)

-- Total number of duplicate records by entire record in source
-- total_source_duplicate1 = group source_duplicate all;

10. total_source_duplicate = foreach (group source_duplicate all) generate CONCAT('Total Number of Duplicate Records in Source (by entire row): ','
',(chararray)SUM(source_duplicate.no_of_records)) as
no_of_duplicate_records_in_source_by_row;

10-1. dump total_source_duplicate;
OUTPUT:
(Total Number of Duplicate Records in Source (by entire row): 2)

-- Split duplicate and unique records by entire record in target
-- grpd_tgt_by_all_columns = group target by ($INPUT_TARGET_FIELDS);

11. grpd_tgt_all_columns_recordset_with_count = foreach (group target by (segment_id,engagement_id,segment_cd,segment_desc,ey_segment_group)) generate target, COUNT(target) as no_of_records;

11-1. dump grpd_tgt_all_columns_recordset_with_count;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group55)},1)
({(6,200,cd2,desc2,group2)},1)
({(7,100,cd1,desc1,group1)},1)
-- splits unique and duplicate records from the above bag

12. split grpd_tgt_all_columns_recordset_with_count into target_unique if no_of_records == 1, target_duplicate if no_of_records > 1;

12-1. dump target_unique;
OUTPUT:
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)

({(5,500,cd5,desc5,group55)},1)
({(6,200,cd2,desc2,group2)},1)
({(7,100,cd1,desc1,group1)},1)
-- extracts the unique records in target

13. target_unique_records = foreach target_unique generate FLATTEN(target);

13-1. dump target_unique_records;
OUTPUT:
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group55)
(6,200,cd2,desc2,group2)
(7,100,cd1,desc1,group1)

-- extracts the duplicate records in target

14. target_duplicate_records = foreach target_duplicate generate FLATTEN(target);

14-1. dump target_duplicate_records;
OUTPUT:
(1,100,cd1,desc1,group1)
(1,100,cd1,desc1,group1)

-- Total number of unique records by entire record in target
-- total_target_unique1 = group target_unique all;

15. total_target_unique = foreach (group target_unique all) generate CONCAT('Total Number of Unique Records in Target (by entire row): ',' ',(chararray)SUM(target_unique.no_of_records)) as no_of_unique_records_in_target_by_row;

15-1. dump total_target_unique;
OUTPUT:
(Total Number of Unique Records in Target (by entire row): 6)

-- Total number of duplicate records by entire record in target
-- total_target_duplicate1 = group target_duplicate all;

16. total_target_duplicate = foreach (group target_duplicate all) generate CONCAT('Total Number of Duplicate Records in Target (by entire row): ','
',(chararray)SUM(target_duplicate.no_of_records)) as
no_of_duplicate_records_in_target_by_row;

16-1. dump total_target_duplicate;
OUTPUT:

(Total Number of Duplicate Records in Target (by entire row): 2)

-- Split duplicate and unique records by key(s) in source

17. grpd_src_by_keys = group source by (segment_id);

17-1. dump grpd_src_by_keys;
OUTPUT:
(1,{(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)})
(2,{(2,200,cd2,desc2,group2)})
(3,{(3,300,cd3,desc3,group3)})
(4,{(4,400,cd4,desc4,group4)})
(5,{(5,500,cd5,desc5,group5)})
(6,{(6,200,cd2,desc2,group2)})

18. grpd_src_keys_recordset_with_count = foreach grpd_src_by_keys generate source, COUNT(source) as no_of_records;

18-1. dump grpd_src_keys_recordset_with_count;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group5)},1)
({(6,200,cd2,desc2,group2)},1)

19. split grpd_src_keys_recordset_with_count into source_unique_by_keys if no_of_records == 1, source_duplicate_by_keys if no_of_records > 1;

19-1. dump source_unique_by_keys;
OUTPUT:
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group5)},1)
({(6,200,cd2,desc2,group2)},1)

19-2. dump source_duplicate_by_keys;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)

20. source_unique_records_by_keys = foreach source_unique_by_keys generate FLATTEN(source);
20-1. dump source_unique_records_by_keys;

OUTPUT:
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group5)
(6,200,cd2,desc2,group2)

21. source_duplicate_records_by_keys = foreach source_duplicate_by_keys generate FLATTEN(source);
21-1 dump source_duplicate_records_by_keys;
OUTPUT:
(1,100,cd1,desc1,group1)
(1,100,cd1,desc1,group1)

-- Total number of unique records by key(s) in source
-- total_source_unique1_by_keys = group source_unique_by_keys all;

22. total_source_unique_by_keys = foreach (group source_unique_by_keys all) generate CONCAT('Total Number of Unique Records in Source (by key(s)): ',' ',(chararray)SUM(source_unique_by_keys.no_of_records)) as no_of_unique_records_in_source_by_keys;

22-1. dump total_source_unique_by_keys;
OUTPUT:
(Total Number of Unique Records in Source (by key(s)): 5)

-- Total number of duplicate records by entire record in source
-- total_source_duplicate1_by_keys = group source_duplicate_by_keys all;

23. total_source_duplicate_by_keys = foreach (group source_duplicate_by_keys all) generate CONCAT('Total Number of Duplicate Records in Source (by key(s)): ',' ',(chararray)SUM(source_duplicate_by_keys.no_of_records)) as no_of_duplicate_records_in_source_by_keys;

23-1. dump total_source_duplicate_by_keys;
OUTPUT:
(Total Number of Duplicate Records in Source (by key(s)): 2)

-- Split duplicate and unique records by key(s) in target

24. grpd_tgt_by_keys = group target by (segment_id);

24-1. dump grpd_tgt_by_keys;
OUTPUT:
(1,{(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)})
(2,{(2,200,cd2,desc2,group2)})

(3,{(3,300,cd3,desc3,group3)})
(4,{(4,400,cd4,desc4,group4)})
(5,{(5,500,cd5,desc5,group55)})
(6,{(6,200,cd2,desc2,group2)})
(7,{(7,100,cd1,desc1,group1)})

25. grpd_tgt_keys_recordset_with_count = foreach grpd_tgt_by_keys generate target, COUNT(target) as no_of_records;

25-1. dump grpd_tgt_keys_recordset_with_count;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group55)},1)
({(6,200,cd2,desc2,group2)},1)
({(7,100,cd1,desc1,group1)},1)

26. split grpd_tgt_keys_recordset_with_count into target_unique_by_keys if no_of_records == 1, target_duplicate_by_keys if no_of_records > 1;
26-1. dump target_unique_by_keys;
OUTPUT:
({(2,200,cd2,desc2,group2)},1)
({(3,300,cd3,desc3,group3)},1)
({(4,400,cd4,desc4,group4)},1)
({(5,500,cd5,desc5,group55)},1)
({(6,200,cd2,desc2,group2)},1)
({(7,100,cd1,desc1,group1)},1)

26-2. dump target_duplicate_by_keys;
OUTPUT:
({(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},2)

27. target_unique_records_by_keys = foreach target_unique_by_keys generate FLATTEN(target);

27-1. dump target_unique_records_by_keys;
OUTPUT:
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group55)
(6,200,cd2,desc2,group2)
(7,100,cd1,desc1,group1)

28. target_duplicate_records_by_keys = foreach target_duplicate_by_keys generate FLATTEN(target);

28-1. dump target_duplicate_records_by_keys;
OUTPUT:
(1,100,cd1,desc1,group1)
(1,100,cd1,desc1,group1)

-- Total number of unique records by key(s) in target
-- total_target_unique1_by_keys = group target_unique_by_keys all;
29. total_target_unique_by_keys = foreach (group target_unique_by_keys all) generate CONCAT('Total Number of Unique Records in Target (by key(s)): ',' ',(chararray)SUM(target_unique_by_keys.no_of_records)) as no_of_unique_records_in_target_by_keys;
29-1. dump total_target_unique_by_keys;
OUTPUT:
(Total Number of Unique Records in Target (by key(s)): 6)

-- Total number of duplicate records by key(s) in target
-- total_target_duplicate1_by_keys = group target_duplicate_by_keys all;

30. total_target_duplicate_by_keys = foreach (group target_duplicate_by_keys all) generate CONCAT('Total Number of Duplicate Records in Target (by key(s)): ',' ',(chararray)SUM(target_duplicate_by_keys.no_of_records)) as no_of_duplicate_records_in_target_by_keys;

30-1. dump total_target_duplicate_by_keys;
OUTPUT:
(Total Number of Duplicate Records in Target (by key(s)): 2)

-- Full Outer join of source and target data (grouped distinct data)

31. joined_distinct_source_target_by_keys = join grpd_src_by_keys by group FULL OUTER, grpd_tgt_by_keys by group;

31-1. dump joined_distinct_source_target_by_keys;
OUTPUT:
(1,{(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},1,{(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)})
(2,{(2,200,cd2,desc2,group2)},2,{(2,200,cd2,desc2,group2)})
(3,{(3,300,cd3,desc3,group3)},3,{(3,300,cd3,desc3,group3)})
(4,{(4,400,cd4,desc4,group4)},4,{(4,400,cd4,desc4,group4)})
(5,{(5,500,cd5,desc5,group5)},5,{(5,500,cd5,desc5,group55)})
(6,{(6,200,cd2,desc2,group2)},6,{(6,200,cd2,desc2,group2)})
(,,7,{(7,100,cd1,desc1,group1)})

-- Data(Key(s)) found in both source and target

32. common_data_by_keys = filter joined_distinct_source_target_by_keys by grpd_src_by_keys::source IS NOT NULL and grpd_tgt_by_keys::target IS NOT NULL;

32-1. dump common_data_by_keys;
OUTPUT:
(1,{(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)},1,{(1,100,cd1,desc1,group1),(1,100,cd1,desc1,group1)})
(2,{(2,200,cd2,desc2,group2)},2,{(2,200,cd2,desc2,group2)})
(3,{(3,300,cd3,desc3,group3)},3,{(3,300,cd3,desc3,group3)})
(4,{(4,400,cd4,desc4,group4)},4,{(4,400,cd4,desc4,group4)})
(5,{(5,500,cd5,desc5,group5)},5,{(5,500,cd5,desc5,group55)})
(6,{(6,200,cd2,desc2,group2)},6,{(6,200,cd2,desc2,group2)})

33. common_data_in_source_by_keys = foreach common_data_by_keys generate FLATTEN(grpd_src_by_keys::source);

33-1. dump common_data_in_source_by_keys;
OUTPUT:
(1,100,cd1,desc1,group1)
(1,100,cd1,desc1,group1)
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group5)
(6,200,cd2,desc2,group2)

34. common_data_in_target_by_keys = foreach common_data_by_keys generate FLATTEN(grpd_tgt_by_keys::target);

34-1. dump common_data_in_target_by_keys;
OUTPUT:
(1,100,cd1,desc1,group1)
(1,100,cd1,desc1,group1)
(2,200,cd2,desc2,group2)
(3,300,cd3,desc3,group3)
(4,400,cd4,desc4,group4)
(5,500,cd5,desc5,group55)
(6,200,cd2,desc2,group2)

-- Data(Key(s)) found in source and not in target

35. data_not_in_target_by_keys = filter joined_distinct_source_target_by_keys by grpd_src_by_keys::source IS NOT NULL and grpd_tgt_by_keys::target IS NULL;

35-1. dump data_not_in_target_by_keys;
OUTPUT:

36. data_not_in_target_by_keys1 = foreach data_not_in_target_by_keys generate
FLATTEN(grpd_src_by_keys::source);

36-1. dump data_not_in_target_by_keys1;
OUTPUT:

-- Data(Key(s)) found in target and not in source

37. data_not_in_source_by_keys = filter joined_distinct_source_target_by_keys by
grpd_src_by_keys::source IS NULL and grpd_tgt_by_keys::target IS NOT NULL;

37-1. dump data_not_in_source_by_keys;
OUTPUT:
(,,7,{(7,100,cd1,desc1,group1)})

38. data_not_in_source_by_keys1 = foreach data_not_in_source_by_keys generate
FLATTEN(grpd_tgt_by_keys::target);

38-1. dump data_not_in_source_by_keys1;
OUTPUT:
(7,100,cd1,desc1,group1)

-- Number of Key(s) matching in Source and Target
-- no_of_matching_keys = group common_data_by_keys all;

39. no_of_matching_keys1 = foreach (group common_data_by_keys all) generate
CONCAT('Total Number of Key(s) matching in Source and Target: ','
',(chararray)COUNT(common_data_by_keys)) as unique_matching_keys;

39-1. dump no_of_matching_keys1;
OUTPUT:(Total Number of Key(s) matching in Source and Target: 6)

--Number of Records in Source with matching Key(s)
-- no_of_recs_matching_keys_in_src1 = group common_data_in_source_by_keys all;

40. no_of_recs_matching_keys_in_src = foreach (group common_data_in_source_by_keys all)
generate CONCAT('Total Number of records in source with matching key(s): ','
',(chararray)COUNT(common_data_in_source_by_keys)) as no_src_recs_matching_keys;

40-1. dump no_of_recs_matching_keys_in_src;
OUTPUT:
(Total Number of records in source with matching key(s): 7)

--Number of Records in Target with matching Key(s)
-- no_of_recs_matching_keys_in_tgt1 = group common_data_in_target_by_keys all;

41. no_of_recs_matching_keys_in_tgt = foreach (group common_data_in_target_by_keys all) generate CONCAT('Total Number of records in target with matching key(s): ',' ',(chararray)COUNT(common_data_in_target_by_keys)) as no_tgt_recs_matching_keys;

41-1. dump no_of_recs_matching_keys_in_tgt;
OUTPUT:
(Total Number of records in target with matching key(s): 7)

-- Number of Key(s) in source missing in target
-- no_of_keys_in_src_not_in_tgt1 = group data_not_in_target_by_keys all;

42. no_of_keys_in_src_not_in_tgt = foreach (group data_not_in_target_by_keys all) generate CONCAT('Total Number of Key(s) in source missing in target: ',' ',(chararray)COUNT(data_not_in_target_by_keys.grpd_src_by_keys::source)) as no_of_keys_in_src_not_in_tgt;

42-1. dump no_of_keys_in_src_not_in_tgt;
OUTPUT:

-- Number of records in source with keys(s) not in target
-- no_of_recs_keys_in_src_not_in_tgt1 = group data_not_in_target_by_keys1 all;

43. no_of_recs_keys_in_src_not_in_tgt = foreach (group data_not_in_target_by_keys1 all) generate CONCAT('Total Number of records in source with Key(s) missing in target: ',' ',(chararray)COUNT(data_not_in_target_by_keys1)) as no_src_recs_keys_not_in_tgt;

43-1. dump no_of_recs_keys_in_src_not_in_tgt;
OUTPUT:

-- Number of Key(s) in target missing in source
-- no_of_keys_in_tgt_not_in_src1 = group data_not_in_source_by_keys all;

44. no_of_keys_in_tgt_not_in_src = foreach (group data_not_in_source_by_keys all) generate CONCAT('Total Number of Key(s) in target missing in source: ',' ',(chararray)COUNT(data_not_in_source_by_keys.grpd_tgt_by_keys::target)) as no_of_keys_in_tgt_not_in_src;

44-1. dump no_of_keys_in_tgt_not_in_src;
OUTPUT:
(Total Number of Key(s) in target missing in source: 1)

-- Number of records in target with keys(s) not in source
-- no_of_recs_keys_in_tgt_not_in_src1 = group data_not_in_source_by_keys1 all;

45. no_of_recs_keys_in_tgt_not_in_src = foreach (group data_not_in_source_by_keys1 all) generate CONCAT('Total Number of records in target with Key(s) missing in source: ',' ',(chararray)COUNT(data_not_in_source_by_keys1)) as no_tgt_recs_keys_not_in_src;

45-1. dump no_of_recs_keys_in_tgt_not_in_src;
OUTPUT:
(Total Number of records in target with Key(s) missing in source: 1)

--All stat information in a single alias

46. stats = UNION
total_records_src,total_records_tgt,total_source_unique,total_source_duplicate,
total_target_unique,total_target_duplicate, total_source_unique_by_keys,
total_source_duplicate_by_keys, total_target_unique_by_keys, total_target_duplicate_by_keys,
no_of_matching_keys1, no_of_recs_matching_keys_in_src, no_of_recs_matching_keys_in_tgt,
no_of_keys_in_src_not_in_tgt, no_of_recs_keys_in_src_not_in_tgt,
no_of_keys_in_tgt_not_in_src, no_of_recs_keys_in_tgt_not_in_src;

46-1. dump stats;
OUTPUT:
(Total Number of records in target with Key(s) missing in source: 1)
(Total Number of Duplicate Records in Source (by entire row): 2)
(Total Number of Duplicate Records in Target (by entire row): 2)
(Total Number of Unique Records in Source (by entire row): 5)
(Total Number of Unique Records in Target (by entire row): 6)
(Total Number of Duplicate Records in Source (by key(s)): 2)
(Total Number of Duplicate Records in Target (by key(s)): 2)
(Total Number of records in source with matching key(s): 7)
(Total Number of records in target with matching key(s): 7)
(Total Number of Unique Records in Source (by key(s)): 5)
(Total Number of Unique Records in Target (by key(s)): 6)
(Total Number of Key(s) matching in Source and Target: 6)
(Total Number of Key(s) in target missing in source: 1)
(Total Number of Records in Source: 7)
(Total Number of Records in Target: 8)

47. store stats into '/user/himanshu/hadoopqatstool/output/19-11-2015-13-19/statistics' using PigStorage(',');