tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases.

This is a brief tutorial that explains how to make use of Sqoop in Hadoop ecosystem.

Audience

This tutorial is prepared for professionals aspiring to make a career in Big Data Analytics using Hadoop Framework with Sqoop. ETL developers and professionals who are into analytics in general may as well use this tutorial to good effect.

Prerequisites

Before proceeding with this tutorial, you need a basic knowledge of Core Java, Database concepts of SQL, Hadoop File system, and any of Linux operating system flavors.

Copyright & Disclaimer

Table of Contents

1. INTRODUCTION

The traditional application management system, that is, the interaction of applications with relational database using RDBMS, is one of the sources that generate Big Data. Such Big Data, generated by RDBMS, is stored in **Relational Database Servers** in the relational database structure.

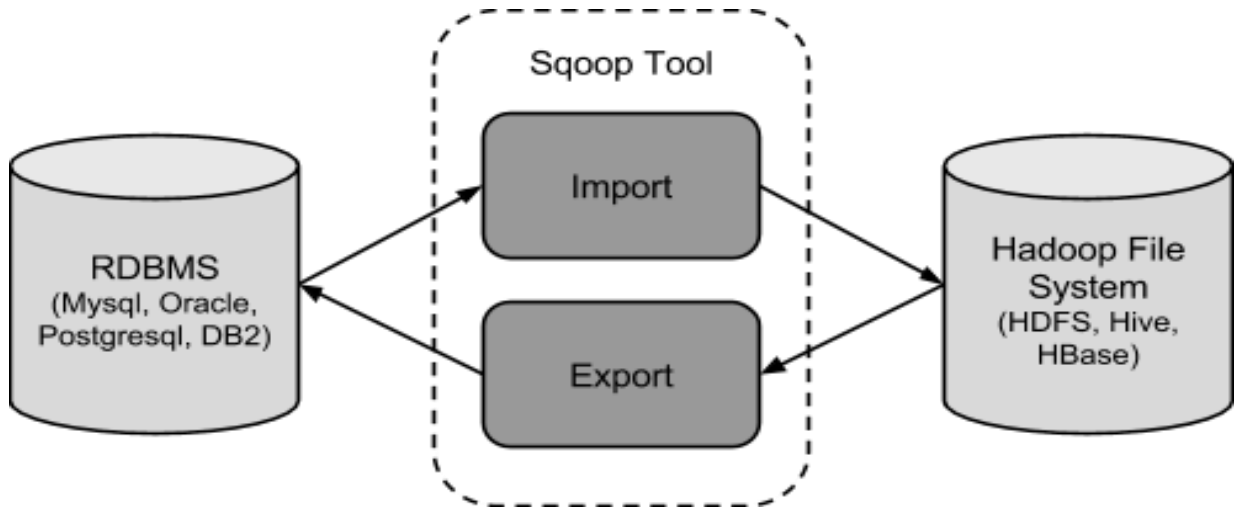
When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. of the Hadoop ecosystem came into picture, they required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them. Here, Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop's HDFS.

Sqoop: "SQL to Hadoop and Hadoop to SQL"

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. It is provided by the Apache Software Foundation.

How Sqoop Works?

The following image describes the workflow of Sqoop.



Sqoop Import

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

2. INSTALLATION

As Sqoop is a sub-project of Hadoop, it can only work on Linux operating system. Follow the steps given below to install Sqoop on your system.

Step 1: Verifying JAVA Installation

You need to have Java installed on your system before installing Sqoop. Let us verify Java installation using the following command:

```
$ java –version
```

If Java is already installed on your system, you get to see the following response:

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

If Java is not installed on your system, then follow the steps given below.

Installing Java

Follow the simple steps given below to install Java on your system.

Step 1

Download Java (JDK <latest version> - X64.tar.gz) by visiting the following link http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html.

Then jdk-7u71-linux-x64.tar.gz will be downloaded onto your system.

Step 2

Generally, you can find the downloaded Java file in the Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

```
$ cd Downloads/
$ ls
jdk-7u71-linux-x64.gz
$ tar zxf jdk-7u71-linux-x64.gz
```

```
$ ls
jdk1.7.0_71   jdk-7u71-linux-x64.gz
```

Step 3

To make Java available to all the users, you have to move it to the location "/usr/local/". Open root, and type the following commands.

```
$ su
password:

# mv jdk1.7.0_71 /usr/local/java
# exitStep IV:
```

Step 4

For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/java
export PATH=PATH:$JAVA_HOME/bin
```

Now apply all the changes into the current running system.

```
$ source ~/.bashrc
```

Step 5

Use the following commands to configure Java alternatives:

```
# alternatives --install /usr/bin/java java usr/local/java/bin/java 2
# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2
# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2

# alternatives --set java usr/local/java/bin/java
# alternatives --set javac usr/local/java/bin/javac
# alternatives --set jar usr/local/java/bin/jar
```

Now verify the installation using the command **java -version** from the terminal as explained above.

Step 2: Verifying Hadoop Installation

Hadoop must be installed on your system before installing Sqoop. Let us verify the Hadoop installation using the following command:

```
$ hadoop version
```

If Hadoop is already installed on your system, then you will get the following response:

```
Hadoop 2.4.1
--
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
```

If Hadoop is not installed on your system, then proceed with the following steps:

Downloading Hadoop

Download and extract Hadoop 2.4.1 from Apache Software Foundation using the following commands.

```
$ su
password:

# cd /usr/local
# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/
hadoop-2.4.1.tar.gz
# tar xzf hadoop-2.4.1.tar.gz
# mv hadoop-2.4.1/* to hadoop/
# exit
```

Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 2.4.1 in pseudo-distributed mode.

Step 1: Setting up Hadoop

You can set Hadoop environment variables by appending the following commands to ~/.bashrc file.

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Now, apply all the changes into the current running system.

```
$ source ~/.bashrc
```

Step 2: Hadoop Configuration

You can find all the Hadoop configuration files in the location "$HADOOP_HOME/etc/hadoop". You need to make suitable changes in those configuration files according to your Hadoop infrastructure.

```
$ cd $HADOOP_HOME/etc/hadoop
```

In order to develop Hadoop programs using java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing JAVA_HOME value with the location of java in your system.

```
export JAVA_HOME=/usr/local/java
```

Given below is the list of files that you need to edit to configure Hadoop.

core-site.xml

The core-site.xml file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and the size of Read/Write buffers.

Open the core-site.xml and add the following properties in between the <configuration> and </configuration> tags.

```
<configuration>
 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:9000</value>
 </property>
```

```
</configuration>
```

hdfs-site.xml

The hdfs-site.xml file contains information such as the value of replication data, namenode path, and datanode path of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

```
dfs.replication (data replication value) = 1

(In the following path /hadoop/ is the user name.
hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)
namenode path = //home/hadoop/hadoopinfra/hdfs/namenode

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)
datanode path = //home/hadoop/hadoopinfra/hdfs/datanode
```

Open this file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>
   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>
   <property>
      <name>dfs.name.dir</name>
      <value>file:///home/hadoop/hadoopinfra/hdfs/namenode</value>
   </property>
   <property>
      <name>dfs.data.dir</name>
      <value>file:///home/hadoop/hadoopinfra/hdfs/datanode</value>
   </property>
</configuration>
```

Note: In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

yarn-site.xml

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>
   <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
   </property>
</configuration>
```

mapred-site.xml

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, you need to copy the file from mapred-site.xml.template to mapred-site.xml file using the following command.

```
$ cp mapred-site.xml.template mapred-site.xml
```

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>
     <property>
          <name>mapreduce.framework.name</name>
          <value>yarn</value>
     </property>
</configuration>
```

Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

Step 1: Name Node Setup

Set up the namenode using the command "hdfs namenode -format" as follows.

```
$ cd ~
$ hdfs namenode -format
```

The expected result is as follows.

```
10/24/14 21:30:55 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = localhost/192.168.1.11
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.4.1
...
...
10/24/14 21:30:56 INFO common.Storage: Storage directory
/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.
10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to
retain 1 images with txid >= 0
10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0
10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at localhost/192.168.1.11
************************************************************/
```

Step 2: Verifying Hadoop dfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

```
$ start-dfs.sh
```

The expected output is as follows:

```
10/24/14 21:37:56
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/hadoop-
2.4.1/logs/hadoop-hadoop-namenode-localhost.out
localhost: starting datanode, logging to /home/hadoop/hadoop-
```

```
2.4.1/logs/hadoop-hadoop-datanode-localhost.out
Starting secondary namenodes [0.0.0.0]
```

Step 3: Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

```
$ start-yarn.sh
```

The expected output is as follows:

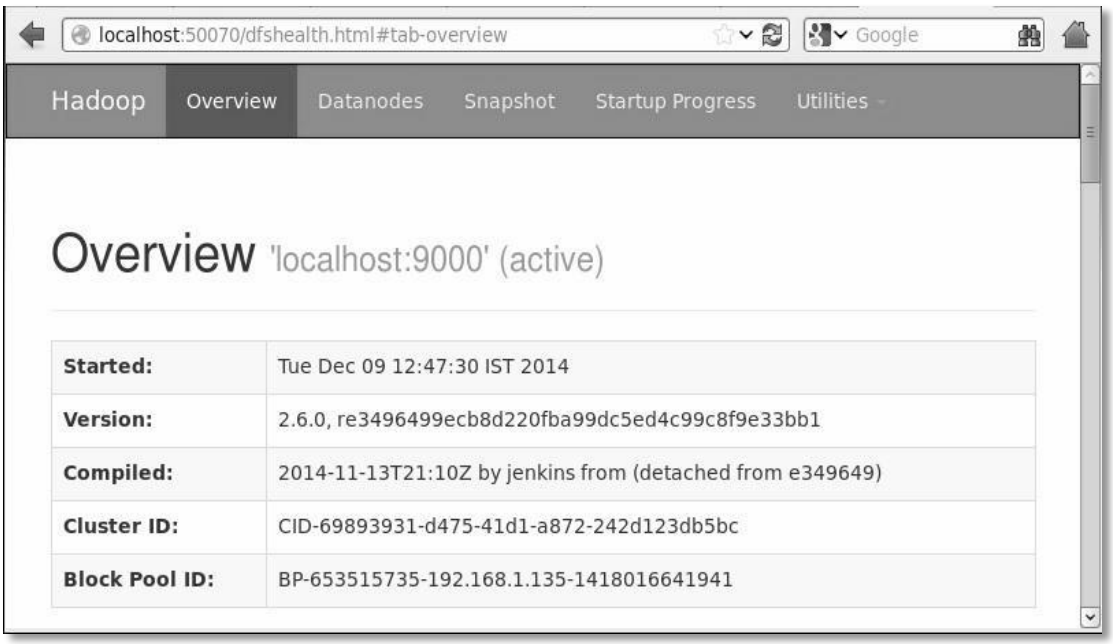
```
starting yarn daemons
starting resourcemanager, logging to /home/hadoop/hadoop-
2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out
localhost: starting node manager, logging to /home/hadoop/hadoop-
2.4.1/logs/yarn-hadoop-nodemanager-localhost.out
```

Step 4: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following URL to get Hadoop services on your browser.

```
http://localhost:50070/
```

The following image depicts a Hadoop browser.

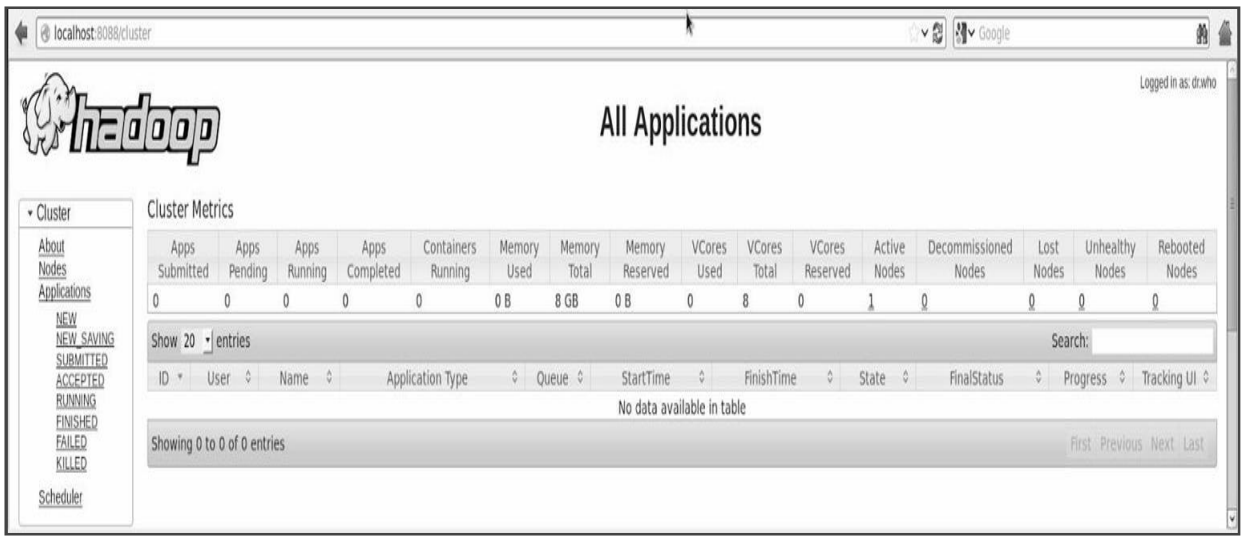


Step 5: Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

```
http://localhost:8088/
```

The following image depicts the Hadoop cluster browser.



Step 3: Downloading Sqoop

We can download the latest version of Sqoop from the following link http://mirrors.ibiblio.org/apache/sqoop/1.4.5/. For this tutorial, we are using version 1.4.5, that is, **sqoop-1.4.5.bin__hadoop-2.0.4-alpha.tar.gz**.

Step 4: Installing Sqoop

The following commands are used to extract the Sqoop tar ball and move it to "/usr/lib/sqoop" directory.

```
$tar -xvf sqoop-1.4.4.bin__hadoop-2.0.4-alpha.tar.gz
$ su
password:

# mv sqoop-1.4.4.bin__hadoop-2.0.4-alpha /usr/lib/sqoop
#exit
```

Step 5: Configuring bashrc

You have to set up the Sqoop environment by appending the following lines to **~/.bashrc** file:

```
#Sqoop

export SQOOP_HOME=/usr/lib/sqoop
export PATH=$PATH:$SQOOP_HOME/bin
```

The following command is used to execute **~/.bashrc** file.

```
$ source ~/.bashrc
```

Step 6: Configuring Sqoop

To configure Sqoop with Hadoop, you need to edit the **sqoop-env.sh** file, which is placed in the **$SQOOP_HOME/conf** directory. First of all, Redirect to Sqoop config directory and copy the template file using the following command:

```
$ cd $SQOOP_HOME/conf

$ mv sqoop-env-template.sh sqoop-env.sh
```

Open **sqoop-env.sh** and edit the following lines:

```
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

Step 7: Download and Configure mysql-connector-java

We can download **mysql-connector-java-5.1.30.tar.gz** file from the following link http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/.

The following commands are used to extract mysql-connector-java tarball and move **mysql-connector-java-5.1.30-bin.jar** to /usr/lib/sqoop/lib directory.

```
$ tar -zxf mysql-connector-java-5.1.30.tar.gz

$ su

password:


# cd mysql-connector-java-5.1.30

# mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib
```

Step 8: Verifying Sqoop

The following command is used to verify the Sqoop version.

```
$ cd $SQOOP_HOME/bin
$ sqoop-version
```

Expected output:

```
14/12/17 14:52:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
Sqoop 1.4.5 git commit id 5b34accaca7de251fc91161733f906af2eddbe83
Compiled by abe on Fri Aug 1 11:19:26 PDT 2014
```

Sqoop installation is complete.

3. IMPORT

This chapter describes how to import data from MySQL database to Hadoop HDFS. The 'Import tool' imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in the text files or as binary data in Avro and Sequence files.

Syntax

The following syntax is used to import data into HDFS.

```
$ sqoop import (generic-args) (import-args)
$ sqoop-import (generic-args) (import-args)
```

Example

Let us take an example of three tables named as **emp**, **emp_add**, and **emp_contact**, which are in a database called **userdb** in a MySQL database server.

The three tables and their data are as follows.

emp:

id	name	deg	salary	dept
1201	gopal	manager	50,000	TP
1202	manisha	Proof reader	50,000	TP
1203	khalil	php dev	30,000	AC
1204	prasanth	php dev	30,000	AC
1204	kranthi	admin	20,000	TP

emp_add:

id	hno	street	city
1201	288A	vgiri	jublee
1202	108I	aoc	sec-bad
1203	144Z	pgutta	hyd
1204	78B	old city	sec-bad
1205	720X	hitec	sec-bad

emp_contact:

id	phno	email
1201	2356742	gopal@tp.com
1202	1661663	manisha@tp.com
1203	8887776	khalil@ac.com
1204	9988774	prasanth@ac.com
1205	1231231	kranthi@tp.com

Importing a Table

Sqoop tool 'import' is used to import table data from the table to the Hadoop file system as a text file or a binary file.

The following command is used to import the **emp** table from MySQL database server to HDFS.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp --m 1
```

If it is executed successfully, then you get the following output.

```
14/12/22 15:24:54 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5

14/12/22 15:24:56 INFO manager.MySQLManager: Preparing to use a MySQL
streaming resultset.


14/12/22 15:24:56 INFO tool.CodeGenTool: Beginning code generation

14/12/22 15:24:58 INFO manager.SqlManager: Executing SQL statement: SELECT
t.* FROM `emp` AS t LIMIT 1


14/12/22 15:24:58 INFO manager.SqlManager: Executing SQL statement: SELECT
t.* FROM `emp` AS t LIMIT 1


14/12/22 15:24:58 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is
/usr/local/hadoop

14/12/22 15:25:11 INFO orm.CompilationManager: Writing jar file:
/tmp/sqoop-hadoop/compile/cebe706d23ebb1fd99c1f063ad51ebd7/emp.jar

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

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

14/12/22 15:25:40 INFO mapreduce.Job: The url to track the job:
http://localhost:8088/proxy/application_1419242001831_0001/

14/12/22 15:26:45 INFO mapreduce.Job: Job job_1419242001831_0001 running in
uber mode : false

14/12/22 15:26:45 INFO mapreduce.Job:  map 0% reduce 0%

14/12/22 15:28:08 INFO mapreduce.Job:  map 100% reduce 0%

14/12/22 15:28:16 INFO mapreduce.Job: Job job_1419242001831_0001 completed
successfully

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

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

14/12/22 15:28:17 INFO mapreduce.ImportJobBase: Transferred 145 bytes in
177.5849 seconds (0.8165 bytes/sec)

14/12/22 15:28:17 INFO mapreduce.ImportJobBase: Retrieved 5 records.
```

To verify the imported data in HDFS, use the following command.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*
```

It shows you the **emp** table data and fields are separated with comma (,).

```
1201, gopal,   manager, 50000, TP
1202, manisha, preader, 50000, TP
1203, kalil,   php dev, 30000, AC
1204, prasanth,php dev, 30000, AC
1205, kranthi, admin,   20000, TP
```

Importing into Target Directory

We can specify the target directory while importing table data into HDFS using the Sqoop import tool.

Following is the syntax to specify the target directory as option to the Sqoop import command.

```
--target-dir <new or exist directory in HDFS>
```

The following command is used to import **emp_add** table data into '/queryresult' directory.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp_add \
--m 1 \
--target-dir /queryresult
```

The following command is used to verify the imported data in /queryresult directory form **emp_add** table.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /queryresult/part-m-*
```

It will show you the emp_add table data with comma (,) separated fields.

```
1201, 288A, vgiri, jublee
1202, 108I, aoc, sec-bad
1203, 144Z, pgutta, hyd
1204, 78B,  oldcity, sec-bad
1205, 720C, hitech, sec-bad
```

Import Subset of Table Data

We can import a subset of a table using the 'where' clause in Sqoop import tool. It executes the corresponding SQL query in the respective database server and stores the result in a target directory in HDFS.

The syntax for where clause is as follows.

```
--where <condition>
```

The following command is used to import a subset of **emp_add** table data. The subset query is to retrieve the employee id and address, who lives in Secunderabad city.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp_add \
--m 1 \
--where "city ='sec-bad'" \
--target-dir /wherequery
```

The following command is used to verify the imported data in /wherequery directory from the **emp_add** table.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /wherequery/part-m-*
```

It will show you the **emp_add** table data with comma (,) separated fields.

```
1202, 108I, aoc,    sec-bad
1204, 78B,  oldcity,sec-bad
1205, 720C, hitech, sec-bad
```

Incremental Import

Incremental import is a technique that imports only the newly added rows in a table. It is required to add 'incremental', 'check-column', and 'last-value' options to perform the incremental import.

The following syntax is used for the incremental option in Sqoop import command.

```
--incremental <mode>
--check-column <column name>
```

```
--last value <last check column value>
```

Let us assume the newly added data into **emp** table is as follows:

```
1206, satish p, grp des, 20000, GR
```

The following command is used to perform the incremental import in the **emp** table.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp \
--m 1 \
--incremental append \
--check-column id \
-last value 1205
```

The following command is used to verify the imported data from **emp** table to HDFS emp/ directory.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*
```

It shows you the **emp** table data with comma (,) separated fields.

```
1201, gopal,    manager, 50000, TP
1202, manisha,  preader, 50000, TP
1203, kalil,    php dev, 30000, AC
1204, prasanth, php dev, 30000, AC
1205, kranthi,  admin,   20000, TP
1206, satish p, grp des, 20000, GR
```

The following command is used to see the modified or newly added rows from the **emp** table.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*1
```

It shows you the newly added rows to the **emp** table with comma (,) separated fields.

```
1206, satish p, grp des, 20000, GR
```

4. IMPORT-ALL-TABLES

This chapter describes how to import all the tables from the RDBMS database server to the HDFS. Each table data is stored in a separate directory and the directory name is same as the table name.

Syntax

The following syntax is used to import all tables.

```
$ sqoop import-all-tables (generic-args) (import-args)
$ sqoop-import-all-tables (generic-args) (import-args)
```

Example

Let us take an example of importing all tables from the **userdb** database. The list of tables that the database **userdb** contains is as follows.

```
+--------------------+
|      Tables        |
+--------------------+
|      emp           |
|      emp_add       |
|      emp_contact   |
+--------------------+
```

The following command is used to import all the tables from the **userdb** database.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root
```

Note: If you are using the import-all-tables, it is mandatory that every table in that database must have a primary key field.

The following command is used to verify all the table data to the userdb database in HDFS.

```
$ $HADOOP_HOME/bin/hadoop fs -ls
```

It will show you the list of table names in userdb database as directories.

Output

```
drwxr-xr-x   - hadoop supergroup     0 2014-12-22 22:50 _sqoop
drwxr-xr-x   - hadoop supergroup     0 2014-12-23 01:46 emp
drwxr-xr-x   - hadoop supergroup     0 2014-12-23 01:50 emp_add
drwxr-xr-x   - hadoop supergroup     0 2014-12-23 01:52 emp_contact
```

5. EXPORT

This chapter describes how to export data back from the HDFS to the RDBMS database. The target table must exist in the target database. The files which are given as input to the Sqoop contain records, which are called rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

The default operation is to insert all the record from the input files to the database table using the INSERT statement. In update mode, Sqoop generates the UPDATE statement that replaces the existing record into the database.

Syntax

The following is the syntax for the export command.

```
$ sqoop export (generic-args) (export-args)
$ sqoop-export (generic-args) (export-args)
```

Example

Let us take an example of the employee data in file, in HDFS. The employee data is available in **emp_data** file in 'emp/' directory in HDFS. The **emp_data** is as follows.

```
1201, gopal,    manager,  50000,  TP
1202, manisha,  preader,  50000,  TP
1203, kalil,    php dev,  30000,  AC
1204, prasanth, php dev,  30000,  AC
1205, kranthi,  admin,    20000,  TP
1206, satish p, grp des,  20000,  GR
```

It is mandatory that the table to be exported is created manually and is present in the database from where it has to be exported.

The following query is used to create the table 'employee' in mysql command line.

```
$ mysql
mysql> USE db;
mysql> CREATE TABLE employee (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(20),
```

```
    deg VARCHAR(20),
    salary INT,
    dept VARCHAR(10));
```

The following command is used to export the table data (which is in **emp_data** file on HDFS) to the employee table in db database of Mysql database server.

```
$ sqoop export  \
--connect jdbc:mysql://localhost/db \
--username root \
--table employee  \
--export-dir /emp/emp_data
```

The following command is used to verify the table in mysql command line.

```
mysql>select * from employee;
```

If the given data is stored successfully, then you can find the following table of given employee data.

```
+------+--------------+-------------+-------------------+--------+
| Id   |   Name       | Designation |  Salary           | Dept   |
+------+--------------+-------------+-------------------+--------+
| 1201 |   gopal      |   manager   |  50000            |  TP    |
| 1202 |   manisha    |   preader   |  50000            |  TP    |
| 1203 |   kalil      |   php dev   |  30000            |  AC    |
| 1204 |   prasanth   |   php dev   |  30000            |  AC    |
| 1205 |   kranthi    |   admin     |  20000            |  TP    |
| 1206 | satish p     |   grp des   |  20000            |  GR    |
+------+--------------+-------------+-------------------+--------+
```

6. SQOOP JOB

This chapter describes how to create and maintain the Sqoop jobs. Sqoop job creates and saves the import and export commands. It specifies parameters to identify and recall the saved job. This re-calling or re-executing is used in the incremental import, which can import the updated rows from RDBMS table to HDFS.

Syntax

The following is the syntax for creating a Sqoop job.

```
$ sqoop job (generic-args) (job-args)
  [-- [subtool-name] (subtool-args)]

$ sqoop-job (generic-args) (job-args)
  [-- [subtool-name] (subtool-args)]
```

Create Job (–create)

Here we are creating a job with the name **myjob**, which can import the table data from RDBMS table to HDFS. The following command is used to create a job that is importing data from the **employee** table in the **db** database to the HDFS file.

```
$ sqoop job --create myjob \
--import \
--connect jdbc:mysql://localhost/db \
--username root \
--table employee --m 1
```

Verify Job (–list)

'**--list**' argument is used to verify the saved jobs. The following command is used to verify the list of saved Sqoop jobs.

```
$ sqoop job --list
```

It shows the list of saved jobs.

```
Available jobs:
  myjob
```

Inspect Job (–show)

'**--show**' argument is used to inspect or verify particular jobs and their details. The following command and sample output is used to verify a job called **myjob**.

```
$ sqoop job --show myjob
```

It shows the tools and their options, which are used in **myjob**.

```
Job: myjob
 Tool: import
 Options:
 ----------------------------
 direct.import = true
 codegen.input.delimiters.record = 0
 hdfs.append.dir = false
 db.table = employee
 ...
 incremental.last.value = 1206
 ...
```

Execute Job (–exec)

'**--exec**' option is used to execute a saved job. The following command is used to execute a saved job called **myjob**.

```
$ sqoop job --exec myjob
```

It shows you the following output.

```
10/08/19 13:08:45 INFO tool.CodeGenTool: Beginning code generation
...
```

7. CODEGEN

This chapter describes the importance of 'codegen' tool. From the viewpoint of object-oriented application, every database table has one DAO class that contains 'getter' and 'setter' methods to initialize objects. This tool (-codegen) generates the DAO class automatically.

It generates DAO class in Java, based on the Table Schema structure. The Java definition is instantiated as a part of the import process. The main usage of this tool is to check if Java lost the Java code. If so, it will create a new version of Java with the default delimiter between fields.

Syntax

The following is the syntax for Sqoop codegen command.

```
$ sqoop codegen (generic-args) (codegen-args)
$ sqoop-codegen (generic-args) (codegen-args)
```

Example

Let us take an example that generates Java code for the **emp** table in the **userdb** database.

The following command is used to execute the given example.

```
$ sqoop codegen \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp
```

If the command executes successfully, then it will produce the following output on the terminal.

```
14/12/23 02:34:40 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
14/12/23 02:34:41 INFO tool.CodeGenTool: Beginning code generation
…….
14/12/23 02:34:42 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is
/usr/local/hadoop
Note: /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/emp.java
uses or overrides a deprecated API.
```

```
Note: Recompile with -Xlint:deprecation for details.


14/12/23 02:34:47 INFO orm.CompilationManager: Writing jar file:
/tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/emp.jar
```

Verification

Let us take a look at the output. The path, which is in bold, is the location that the Java code of the **emp** table generates and stores. Let us verify the files in that location using the following commands.

```
$ cd /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/
$ ls
emp.class
emp.jar
emp.java
```

If you want to verify in depth, compare the **emp** table in the **userdb** database and **emp.java** in the following directory

`/tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/.`

8. EVAL

This chapter describes how to use the Sqoop 'eval' tool. It allows users to execute user-defined queries against respective database servers and preview the result in the console. So, the user can expect the resultant table data to import. Using eval, we can evaluate any type of SQL query that can be either DDL or DML statement.

Syntax

The following syntax is used for Sqoop eval command.

```
$ sqoop eval (generic-args) (eval-args)
$ sqoop-eval (generic-args) (eval-args)
```

Select Query Evaluation

Using eval tool, we can evaluate any type of SQL query. Let us take an example of selecting limited rows in the **employee** table of **db** database. The following command is used to evaluate the given example using SQL query.

```
$ sqoop eval \
--connect jdbc:mysql://localhost/db \
--username root \
--query "SELECT * FROM employee LIMIT 3"
```

If the command executes successfully, then it will produce the following output on the terminal.

```
+------+--------------+-------------+-------------------+--------+
| Id   |   Name       | Designation |  Salary           | Dept   |
+------+--------------+-------------+-------------------+--------+
| 1201 |   gopal      |   manager   |  50000            |  TP    |
| 1202 |   manisha    |   preader   |  50000            |  TP    |
| 1203 |   khalil      |  php dev   |  30000            |  AC    |
+------+--------------+-------------+-------------------+--------+
```

Insert Query Evaluation

Sqoop eval tool can be applicable for both modeling and defining the SQL statements. That means, we can use eval for insert statements too. The following command is used to insert a new row in the **employee** table of **db** database.

```
$ sqoop eval \
--connect jdbc:mysql://localhost/db \
--username root \
-e “INSERT INTO employee VALUES(1207,‘Raju’,‘UI dev’,15000,‘TP’)”
```

If the command executes successfully, then it will display the status of the updated rows on the console.

Or else, you can verify the employee table on MySQL console. The following command is used to verify the rows of **employee** table of **db** database using select’ query.

```
mysql>
mysql> use db;
mysql> SELECT * FROM employee;

+------+--------------+-------------+-------------------+--------+
| Id   |   Name       | Designation |  Salary           | Dept   |
+------+--------------+-------------+-------------------+--------+
| 1201 |   gopal      |   manager   |  50000            |  TP    |
| 1202 |   manisha    |   preader   |  50000            |  TP    |
| 1203 |   khalil     |   php dev   |  30000            |  AC    |
| 1204 |   prasanth   |   php dev   |  30000            |  AC    |
| 1205 |   kranthi    |   admin     |  20000            |  TP    |
| 1206 |   satish p   |   grp des   |  20000            |  GR    |
| 1207 |   Raju       |   UI dev    |  15000            |  TP    |
+------+--------------+-------------+-------------------+--------+
```

9. LIST-DATABASES

This chapter describes how to list out the databases using Sqoop. Sqoop list-databases tool parses and executes the 'SHOW DATABASES' query against the database server. Thereafter, it lists out the present databases on the server.

Syntax

The following syntax is used for Sqoop list-databases command.

```
$ sqoop list-databases (generic-args) (list-databases-args)
$ sqoop-list-databases (generic-args) (list-databases-args)
```

Sample Query

The following command is used to list all the databases in the MySQL database server.

```
$ sqoop list-databases \
--connect jdbc:mysql://localhost/ \
--username root
```

If the command executes successfully, then it will display the list of databases in your MySQL database server as follows.

```
...
13/05/31 16:45:58 INFO manager.MySQLManager: Preparing to use a MySQL
streaming resultset.


mysql
test
userdb
db
```

10. LIST-TABLES

This chapter describes how to list out the tables of a particular database in MySQL database server using Sqoop. Sqoop list-tables tool parses and executes the 'SHOW TABLES' query against a particular database. Thereafter, it lists out the present tables in a database.

Syntax

The following syntax is used for Sqoop list-tables command.

```
$ sqoop list-tables (generic-args) (list-tables-args)
$ sqoop-list-tables (generic-args) (list-tables-args)
```

Sample Query

The following command is used to list all the tables in the **userdb** database of MySQL database server.

```
$ sqoop list-tables \
--connect jdbc:mysql://localhost/userdb \
--username root
```

If the command is executes successfully, then it will display the list of tables in the **userdb** database as follows.

```
...
13/05/31 16:45:58 INFO manager.MySQLManager: Preparing to use a MySQL
streaming resultset.


emp
emp_add
emp_contact
```