**Sqoop User Guide**

## 1. Introduction

Sqoop is a tool designed to transfer data between Hadoop and relational databases or mainframes. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle or a mainframe into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS.

Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

This document describes how to get started using Sqoop to move data between databases and Hadoop or mainframe to Hadoop and provides reference information for the operation of the Sqoop command-line tool suite. This document is intended for:

* System and application programmers
* System administrators
* Database administrators
* Data analysts
* Data engineers

Operation:

+++++++++++++++++++++++++++++cdh 158++++++++++++++++++++++++++=====

sqoop import \

--connect jdbc:oracle:thin:@172.16.19.71:1521:AADC \

--username kit\_dev \

--password kit\_dev \

--query "select \* from PS\_SAFE\_DUTY where \$CONDITIONS" \

-m 1 \

--fields-terminated-by '\t' \

--null-string '\\N' \

--null-non-string '\\N' \

--as-textfile \

--direct \

--delete-target-dir \

--target-dir /tmp/sqoop\_test1/PS\_SAFE\_DUTY

| **Argument （参数）** | **Description （描述）** |
| --- | --- |
| --append (追加) | Append data to an existing dataset in HDFS |
| --as-avrodatafile (avro格式文件) | Imports data to Avro Data Files |
| --as-sequencefile （sequencefile 文件） | Imports data to SequenceFiles |
| --as-textfile （文本文件） | Imports data as plain text (default) |
| --as-parquetfile （parquet 文件） | Imports data to Parquet Files |
| --boundary-query <statement> | Boundary query to use for creating splits |
| --columns <col,col,col…>（表的列名） | Columns to import from table |
| --delete-target-dir (删除已存在的目录) | Delete the import target directory if it exists |
| --direct | Use direct connector if exists for the database |
| --fetch-size <n> | Number of entries to read from database at once. |
| --inline-lob-limit <n> | Set the maximum size for an inline LOB |
| -m,--num-mappers <n> | Use *n* map tasks to import in parallel |
| -e,--query <statement> | Import the results of *statement*. |
| --split-by <column-name> | Column of the table used to split work units. Cannot be used with --autoreset-to-one-mapper option. |
| --autoreset-to-one-mapper | Import should use one mapper if a table has no primary key and no split-by column is provided. Cannot be used with --split-by <col> option. |
| --table <table-name> (表名) | Table to read |
| --target-dir <dir> （目标文件夹） | HDFS destination dir |
| --warehouse-dir <dir> | HDFS parent for table destination |
| --where <where clause> | WHERE clause to use during import |
| -z,--compress | Enable compression |
| --compression-codec <c> | Use Hadoop codec (default gzip) |
| --null-string <null-string> | The string to be written for a null value for string columns |
| --null-non-string <null-string> | The string to be written for a null value for non-string columns |

For example:

$ sqoop import \

--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \

--split-by a.id --target-dir /user/foo/joinresults

Alternately, the query can be executed once and imported serially, by specifying a single map task with -m 1:

$ sqoop import \

--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \

-m 1 --target-dir /user/foo/joinresults

**Table 8. Hive arguments:**

| **Argument** | **Description** |
| --- | --- |
| --hive-home <dir> | Override $HIVE\_HOME |
| --hive-import | Import tables into Hive (Uses Hive’s default delimiters if none are set.) |
| --hive-overwrite | Overwrite existing data in the Hive table. |
| --create-hive-table | If set, then the job will fail if the target hive |
|  | table exits. By default this property is false. |
| --hive-table <table-name> | Sets the table name to use when importing to Hive. |
| --hive-drop-import-delims | Drops *\n*, *\r*, and *\01* from string fields when importing to Hive. |
| --hive-delims-replacement | Replace *\n*, *\r*, and *\01* from string fields with user defined string when importing to Hive. |
| --hive-partition-key | Name of a hive field to partition are sharded on |
| --hive-partition-value <v> | String-value that serves as partition key for this imported into hive in this job. |
| --map-column-hive <map> | Override default mapping from SQL type to Hive type for configured columns. |

**Table 9. HBase arguments:**

| **Argument** | **Description** |
| --- | --- |
| --column-family <family> | Sets the target column family for the import |
| --hbase-create-table | If specified, create missing HBase tables |
| --hbase-row-key <col> | Specifies which input column to use as the row key |
|  | In case, if input table contains composite |
|  | key, then <col> must be in the form of a |
|  | comma-separated list of composite key |
|  | attributes |
| --hbase-table <table-name> | Specifies an HBase table to use as the target instead of HDFS |
| --hbase-bulkload | Enables bulk loading |

### 7.3. Example Invocations

The following examples illustrate how to use the import tool in a variety of situations.

A basic import of a table named EMPLOYEES in the corp database:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES

A basic import requiring a login:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--username SomeUser -P

Enter password: (hidden)

Selecting specific columns from the EMPLOYEES table:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--columns "employee\_id,first\_name,last\_name,job\_title"

Controlling the import parallelism (using 8 parallel tasks):

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

-m 8

Storing data in SequenceFiles, and setting the generated class name to com.foocorp.Employee:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--class-name com.foocorp.Employee --as-sequencefile

Specifying the delimiters to use in a text-mode import:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--fields-terminated-by '\t' --lines-terminated-by '\n' \

--optionally-enclosed-by '\"'

Importing the data to Hive:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--hive-import

Importing only new employees:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--where "start\_date > '2010-01-01'"

Changing the splitting column from the default:

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--split-by dept\_id

Verifying that an import was successful:

$ hadoop fs -ls EMPLOYEES

Found 5 items

drwxr-xr-x - someuser somegrp 0 2010-04-27 16:40 /user/someuser/EMPLOYEES/\_logs

-rw-r--r-- 1 someuser somegrp 2913511 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00000

-rw-r--r-- 1 someuser somegrp 1683938 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00001

-rw-r--r-- 1 someuser somegrp 7245839 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00002

-rw-r--r-- 1 someuser somegrp 7842523 2010-04-27 16:40 /user/someuser/EMPLOYEES/part-m-00003

$ hadoop fs -cat EMPLOYEES/part-m-00000 | head -n 10

0,joe,smith,engineering

1,jane,doe,marketing

...

Performing an incremental import of new data, after having already imported the first 100,000 rows of a table:

$ sqoop import --connect jdbc:mysql://db.foo.com/somedb --table sometable \

--where "id > 100000" --target-dir /incremental\_dataset --append

An import of a table named EMPLOYEES in the corp database that uses validation to validate the import using the table row count and number of rows copied into HDFS: [More Details](http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#validation)

$ sqoop import --connect jdbc:mysql://db.foo.com/corp \

--table EMPLOYEES --validate

+++++++++++++++=export all table to /user/root hdfs path++++++++++++++

sqoop import-all-tables \

--connect jdbc:oracle:thin:@172.16.19.71:1521:AADC \

--username kit\_dev \

--password kit\_dev \

-m 1 \

sqoop job --create myjob --import-all-tables \

--connect jdbc:oracle:thin:@172.16.19.71:1521:AADC \

--username kit\_dev \

--password kit\_dev \

-m 1 \

++++++++++++++++check list table++++++++++++++++++

sqoop list-tables --connect jdbc:oracle:thin:@172.16.19.71:1521:AADC \

--username kit\_dev \

--password kit\_dev \

## Sqoop: hadooptutorial

**Sqoop-export**

It is nothing but exporting data from HDFS to database. To use ‘**export**‘ command, a table in database should already exist. then only export functionality in sqoop will works.

export command will works in two ways  
1. insert  
2. update

**1. insert:** insert mode will insert the new records from HDFS to RDBMS table.

**command:**



|  |  |
| --- | --- |
| 1  2 | sqoop export --connect jdbc:mysql://localhost/sqoop\_export --table student\_exported --export-dir /sqoop/newstudent/part-m-00000 |

**Note:** if a record already present in the database table with same primary key, then it will raise **MySQLIntegrityConstraintViolationException** exception.

**2. update:** update mode will update the records in the RDBMS from HDFS data.

update mode only update already existing records, it will not insert new records in the RDBMS.

**command:**



|  |  |
| --- | --- |
| 1  2 | sqoop export --connect jdbc:mysql://localhost/sqoop\_export --export-dir /sqoop/emp\_last/part-m-00000 --update-key id |

**Sqoop-JOB**

Job is nothing but to save a sqoop command and excute n times when we require it. The job command allows you to create and work with saved jobs. Saved jobs remember the parameters used to specify a job, so they can be re-executed by invoking the job by its handle.

The following are Sqoop Job commands,

**Create job:**



|  |  |
| --- | --- |
| 1  2 | sqoop job --create myjob -- import --connect jdbc:mysql://localhost/sqoop\_export --username root -P --table student\_exported --target-dir /sqoop/test\_job |

**Note:**  
There should be space between ‘–‘ and ‘import’.

**list jobs:** it will show all the jobs.



|  |  |
| --- | --- |
| 1  2 | sqoop job --list |

**inspect job:** it will show details about the job.



|  |  |
| --- | --- |
| 1  2 | sqoop job –show <jobname> |

**delete job:** it will delete existing job.



|  |  |
| --- | --- |
| 1  2 | sqoop job --delete <jobname> |

**execute job:** it will execute the job.



|  |  |
| --- | --- |
| 1  2 | sqoop job --exec myjob |

**Sqoop-Merge**

Merge command in sqoop is to merge two existing tables in HDFS to single new table in HDFS.

Let us discuss merge functionality with the example:

The below are tables in RDBMS,

**Table 1:** emp\_1



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <strong>emp\_id  emp\_name</strong>  1       emp-one  2       emp-two  3       emp-three  4       emp-four |

**Table 2:** emp\_2



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <strong>emp\_id  emp\_name</strong>  5       emp-five  6       emp-six  7       emp-seven  8       emp-eight |

now,

Importing emp\_1 into HDFS



|  |  |
| --- | --- |
| 1  2 | sqoop import --connect jdbc:mysql://localhost/sqoop\_practice --table emp\_1 --target-dir /sqoop/emp\_1 |

Importing emp\_2 into HDFS



|  |  |
| --- | --- |
| 1  2 | sqoop import --connect jdbc:mysql://localhost/sqoop\_practice --table emp\_2 --target-dir /sqoop/emp\_1 |

After importing emp\_1 and emp\_2 into HDFS, sqoop will generate a jar into our local file system. with help of that jar we will able to merge the two tables in HDFS

**1. create a database and table as shown below**



|  |  |
| --- | --- |
| 1  2  3  4 | create database test;  use test;  create table emp(id int not null primary key, name VARCHAR(20), age int, sal int, city VARCHAR(20),created\_dt DATE); |

**2. inserted 10000 + records as shown below**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | insert into emp values(1,'Ravi',23,4500,'kkd','2012-01-01');  insert into emp values(2,'Siva',24,4500,'hyd','2012-01-01');  insert into emp values(3,'Madhu',29,4500,'bng','2012-01-01');  insert into emp values(4,'Shiva2',25,4500,'kkd','2012-01-01');  insert into emp values(5,'Ravi',24,4500,'kkd','2012-01-01');  insert into emp values(6,'ravi1',27,4500,'kkd1','2012-01-01');  .  .  .  insert into emp values(10009,'ravi2',21,4500,'kkd2','2012-01-01');  insert into emp values(10010,'ravi3',28,4500,'kkd3','2012-01-01'); |

Note: I have used mysql import to load 10000+ records, as below



|  |  |
| --- | --- |
| 1  2  3 | load data local infile '/home/cloudera/Desktop/All/mysql\_sqoop/uniq.csv' into table emp fields terminated by ','  enclosed by '"' lines terminated by '\n' (id, name, age, sal, city, created\_dt); |

**3. Run below sqoop command to import from MYSQL to HDFS location**



|  |  |
| --- | --- |
| 1  2 | $ sqoop import --connect jdbc:mysql://localhost/test --table emp --username root -password cloudera --target-dir /sqoop/empdata/ |

It created 4 Map jobs and stored in part files in HDFS location.

4. Updated some records randomly



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | update emp set created\_dt='2012-02-02' where id =21;  update emp set created\_dt='2012-02-02' where id =31;  update emp set created\_dt='2012-02-02' where id =41;  update emp set created\_dt='2012-02-02' where id =51;  update emp set created\_dt='2012-02-02' where id =61;  update emp set created\_dt='2012-02-02' where id =71;  update emp set created\_dt='2012-02-02' where id =81;  update emp set created\_dt='2012-02-02' where id =91;  update emp set created\_dt='2012-02-02' where id =101;  update emp set created\_dt='2012-02-02' where id =111;  update emp set created\_dt='2012-02-02' where id =121;  update emp set created\_dt='2012-02-02' where id =131;  update emp set created\_dt='2012-02-02' where id =141;  update emp set created\_dt='2012-02-02' where id =151;  update emp set created\_dt='2012-02-02' where id =161; |

**5. Used below command to pic updated records and merged into single file.**



|  |  |
| --- | --- |
| 1  2  3  4 | sqoop import --connect jdbc:mysql://localhost/test --table emp \  --username root -password cloudera --incremental lastmodified --merge-key id --check-column created\_dt \  --target-dir /sqoop/empdata/ |

**Sqoop-import-hive**

sqoop provides direct import into hive.There are two ways to import into hive

**1. Create-hive-table:**it will be used when data already imported into hdfs but not into hive



|  |  |
| --- | --- |
| 1  2 | sqoop create-hive-table --connect jdbc:mysql://localhost/sqoop\_practice --table emp --hive-table hive\_emp |

By above command, sqoop will create an empty table with DB column properties.To load data we need to use hive command as shown below:



|  |  |
| --- | --- |
| 1  2 | hive> load data inpath '/sqoop/emp\_all/part-r-00000' into table hive\_emp; |

**2.hive-import:** It is a direct import from RDBMS to Hive



|  |  |
| --- | --- |
| 1  2 | hive> load data inpath '/sqoop/emp\_all/part-r-00000' into table hive\_emp; |