***Sqoop***

Sqoop is a command-line interface application for transferring data between relational databases and Hadoop.

Apache Sqoop is a tool in [**Hadoop ecosystem**](https://www.edureka.co/blog/hadoop-ecosystem) which is designed to transfer data between [**HDFS**](https://www.edureka.co/blog/hdfs-tutorial) (Hadoop storage) and relational database servers like mysql, Oracle RDB, SQLite, Teradata, Netezza, Postgres etc. Apache Sqoop imports data from relational databases to HDFS, and exports data from HDFS to relational databases. It efficiently transfers bulk data between Hadoop and external datastores such as enterprise data warehouses, relational databases, etc

Additionally, Sqoopis used to import data from external datastores into Hadoop ecosystem’s tools like [**Hive**](https://www.edureka.co/blog/hive-tutorial/) & [**HBase**](https://www.edureka.co/blog/hbase-tutorial).

Sqoop internally converts the command into MapReduce tasks, which are then executed over HDFS. It uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism.

Sqoop = sql + hadoop

mapTasks: by default maptasks run 4 using primary key.

If don’t have any primary key only one map task are run, if mention –m argument

**Apache Sqoop Tutorial: Key Features of Sqoop**

Sqoop provides many salient features like:

1. ***Full Load***: Apache Sqoop can load the whole table by a single command. You can also load all the tables from a database using a single command.
2. ***Incremental*** ***Load***: Apache Sqoop also provides the facility of incremental load where you can load parts of table whenever it is updated.
3. ***Parallel*** ***import/export***: Sqoop uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism.
4. ***Import*** ***results*** ***of*** ***SQL*** ***query***: You can also import the result returned from an SQL query in HDFS.
5. ***Load*** ***data*** ***directly*** ***into*** ***HIVE/HBase***: You can load data directly into ***Apache Hive*** for analysis and also dump your data in HBase, which is a NoSQL database.
6. ***Compression***: You can compress your data by using deflate(gzip) algorithm with –compress argument, or by specifying –compression-codec argument. You can also load compressed table in ***Apache Hive***.
7. ***Connectors*** ***for*** ***all*** ***major*** ***RDBMS*** ***Databases***: Apache Sqoop provides connectors for multiple RDBMS databases, covering almost the entire circumference.
8. ***Kerberos*** ***Security*** ***Integration***: Kerberos is a computer network authentication protocol which works on the basis of ‘tickets’ to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Sqoop supports Kerberos authentication.
9. ***Support*** ***for*** ***Accumulo***: You can also instruct Sqoop to import the table in Accumulo rather than a directory in HDFS.
10. ***By default 4 partions.and 4 map task are ruuning in internal***

**sqoop:**

-P : enter the password while running time insted of specified in commnad

mysql –u root –p cloudera

show databases

use retail\_db

show tables

**Display the data database from mysql:**

sqoop list-databases --connect jdbc:mysql://localhost/ --username root --password cloudera

sqoop list-tables --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera

**Display tables from database:**

sqoop eval --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --query 'select \* from products limit 2';

sqoop eval --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --query 'show tables'

**insertinot data inot database tables:**

sqoopevl --connect jdbc:mysql://localhost/retailer\_db --username root --password cloudera --e "insert into foo values(42,'bar')

**import Command:**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table products --target-dir /user/cloudera/sqoop/products

hadoopdfs -ls /user/cliderea/sqoop/products/

haddopdfs -cat /user/cloudera/sqoop/products/filename

**incrementl append:**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders --incremental append --check-column order\_id --last-value 68883 -m 1

insert into orders values(68884,"2014-07-23 00:00:00",111,"complete");

insert into orders values(68885,"2014-07-23 00:00:00",111,"complete");

**incrementl Modes:**

Sqoop provides an incremental import mode, which can be used to retrieve only rows newer than some previously-imported set of rows.

**Why and When to use lastmodified mode?**  
⇛ lastmodified works on time-stamped data  
⇛ Use this when rows of the source table may be updated  
⇛ And each such update will set the value of a last-modified column to the current timestamp  
⇛ Rows where the check column holds a timestamp more recent than the timestamp specified with --last-value are imported

Note:―  
⇛ Oracle Timestamp format : ‘DD-Mon-YY HH24:MI:SS.FF‘  
⇛ MySQL Timestamp format : ‘YYYY-MM-DD HH:MI:SS‘  
⇛ Sqoop Timestamp format : ‘YYYY-MM-DD HH24:MI:SS.FF‘  
     Specify --last-value in Double Quotes. ie, timestamp in this case.  
⇛ Make sure that many RDBMS are case sensitive (database, table, columns).

**When to use append mode?**  
⇛ Works for numerical data that is incrementing over time, such as auto-increment keys  
⇛ When importing a table where new rows are continually being added with increasing row id values

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders --incremental append --check-column order\_id --last-value 68883 -m 1

insert into orders values(68884,"2014-07-23 00:00:00",111,"complete");

insert into orders values(68885,"2014-07-23 00:00:00",111,"complete");

Consider a table with 3 records which you already imported to hdfs using sqoop

+------+------------+----------+------+------------+

| sid | city | state | rank | rDate |

+------+------------+----------+------+------------+

| 101 | Chicago | Illinois | 1 | 2014-01-25 |

| 101 | Schaumburg | Illinois | 3 | 2014-01-25 |

| 101 | Columbus | Ohio | 7 | 2014-01-25 |

+------+------------+----------+------+------------+

sqoop import --connect jdbc:mysql://localhost:3306/ydb --table yloc --username root -P

Now you have additional records in the table but no updates on existing records

+------+------------+----------+------+------------+

| sid | city | state | rank | rDate |

+------+------------+----------+------+------------+

| 101 | Chicago | Illinois | 1 | 2014-01-25 |

| 101 | Schaumburg | Illinois | 3 | 2014-01-25 |

| 101 | Columbus | Ohio | 7 | 2014-01-25 |

| 103 | Charlotte | NC | 9 | 2013-04-22 |

| 103 | Greenville | SC | 9 | 2013-05-12 |

| 103 | Atlanta | GA | 11 | 2013-08-21 |

+------+------------+----------+------+------------+

Here you should use an --incremental append with --check-column which specifies the column to be examined when determining which rows to import.

sqoop import --connect jdbc:mysql://localhost:3306/ydb --table yloc --username root -P --check-column rank --incremental append --last-value 7

The above code will insert all the new rows based on the last value.

Now we can think of second case where there are updates in rows

+------+------------+----------+------+------------+

| sid | city | state | rank | rDate |

+------+------------+----------+------+------------+

| 101 | Chicago | Illinois | 1 | 2015-01-01 |

| 101 | Schaumburg | Illinois | 3 | 2014-01-25 |

| 101 | Columbus | Ohio | 7 | 2014-01-25 |

| 103 | Charlotte | NC | 9 | 2013-04-22 |

| 103 | Greenville | SC | 9 | 2013-05-12 |

| 103 | Atlanta | GA | 11 | 2013-08-21 |

| 104 | Dallas | Texas | 4 | 2015-02-02 |

| 105 | Phoenix | Arzona | 17 | 2015-02-24 |

+------+------------+----------+------+------------+

Here we use incremental lastmodified where we will fetch all the updated rows based on date.

sqoop import --connect jdbc:mysql://localhost:3306/ydb --table yloc --username root -P --check-column rDate --incremental lastmodified --last-value 2014-01-25 --target-dir yloc/loc

**Sqoop Jobs:**

Job is nothing but to save a sqoop command and execute 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 jobs by its handle.

**create jobs:**

sqoop job --create orders\_job -- import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders --incremental append --check-column order\_id --last-value 68885

**display jobs:**

sqoop job –list

**execute jobs:**

sqoop job --exec orders\_job

**show all jobs:**

sqoop job –show orders\_job

**delete all jobs:**

sqoop job –delete orders\_job

**Importing table into hive:**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table products --target-dir /user/cloudera/sqoop/product\_temp/ --hive-import --as-parquetfile -–delete-target-dir

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders\_temp/ --hive-import --as-parquetfile

**Delimeters:**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders\_temp/ --enclosed-by \";

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders\_temp/ --enclosed-by \" --fields-terminated-by \| --lines-terminated-by \:

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders2 --target-dir /user/cloudera/sqoop/orders/ --enclosed-by \" --fields-terminated-by \| --lines-terminated-by \: --delete-target-dir;

**eval query:**

the main purpose of eval tool is it allows users to quickly run simple SQL queries against a database. Afterwards, results are printed to the console. Also, it allows users to preview their [Sqoop import](https://data-flair.training/blogs/sqoop-import/) queries to ensure they import the data they expect.

**Note:** Basically, for evaluation purpose only, the eval tool is provided. Moreover, we can only use it to verify database connection from within the Sqoop. Also, to test simple queries. Likewise, we can not use Sqoop tool in production workflows.

Examples:

sqoop eval –connect jdbc:mysql://localhost/db –username root –query “SELECT \* FROM employee LIMIT 3”

sqoop eval –connect jdbc:mysql://localhost/db –username root -e “INSERT INTO employee VALUES(1007,‘Gem’,‘UI dev’,15000,‘TP’)”

By default 4 map tasks

By defautle , seperated to each file.

By default, Sqoop will use four tasks in parallel for the export process.

By defalult map tasks in hadoop limit 50

by default stored as a text file

--as-textfile

if table does't have any primary key only one map task created

\* if specify -m 10 without any primary key by default one map task is execute db

**If don’t have any primary key value in table:**

--split-by empid // emp id it is a close to primary key columns,

There is no guaranty equal distribution of the records, may be one map task contains 500 records onther one contains 5000 records

Example: sqoop import --connect jdbc:mysql://localhost/retail\_db --username root –password –table Employee –split-by empid --target-dir /user/cloudera/sqoop/employee

**passing no of map tasks with –delete-target-dir:**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders --target-dir /user/cloudera/sqoop/orders --delete-target-dir --m 1

**wherecluse and file format:**

sqoop import --connect jdbc:mysql://localhost/retailer\_db --username root -P cloudera --table products --as-textfile -m 1 --target-dir /user/cloudera/sqoop/product\_temp/

**Columns Specific:**

sqoop import --connect jdbc:mysql://localhost/retailer\_db --username root -P cloudera --table products --columns "EMP\_NO,DEPT\_NO,FROM\_TODATE,END\_DATE" --as-textfile -m 1 --target-dir /user/cloudera/sqoop/product\_temp/

**Where cluse:**

sqoop import --connect jdbc:mysql://localhost/retailer\_db --username root -P cloudera --table products --where "EMP\_NO > 1021" --as-textfile -m 1 --target-dir /user/cloudera/sqoop/product\_temp/

**Free form query support with --delete-target-dir**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera -query 'select order\_id,order\_status from orders where order\_status=’ COMPLETE’ $CONDITIONS' --target-dir /user/cloudera/sqoop/orders --delete-target-dir -m 1

**without where cluse**

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera -query 'select order\_id,order\_status from orders where $CONDITIONS' --target-dir /user/cloudera/sqoop/orders --delete-target-dir -m 1

sqoop import --connect jdbc:mysql://localhost/retailer\_db --username root -P cloudera --query 'select eid,ename from emp where $CONDITIONS' --as-textfile -m 1 --target-dir /user/cloudera/sqoop/product\_temp/ --delete-target-dir

**Sqoop import supports only below formats.**

--as-avrodatafile Imports data to Avro Data Files

--as-sequencefile Imports data to SequenceFiles

--as-textfile Imports data as plain text (default)

--as-parquetfile Imports data as parquet file (from sqoop 1.4.6 version)

sqoop import --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera -query 'select order\_id,order\_status from orders where order\_status="COMPLETE" and $CONDITIONS' --target-dir /user/cloudera/sqoop/orders --delete-target-dir -m 1 --as-parquetfile

**Export Commands:**

create table product2 like products;

select \* from prodcut2

sqoop export --connect jdbc:mysql://localhost/retail\_db --username root --password cloudera --table orders2 --export-dir /user/cloudera/sqoop/orders

**Export commnad staging:**

import all tables from databases

sqoop import-all-tables --connect jdbc:mysql://localhost/retailer\_db

**Incremental loading:**

insert into pruduct values(1346,30,'colgate',30,25,'test image')

insert into pruduct values(1347,30,'pepesedent',30,25,'test image2')

**only two records import from data base:**

sqoop import --connect jdbc:mysql://localhost/retailer\_db --username root -P cloudera --table products --target-dir /user/cloudera/sqoop/product\_temp/ --incremental append --check-column product\_id --last-value 1345

hdfsdfs -ls /user/cliderea/sqoop/products/

if dot wanto remember count:

Creta job import/export:

sqoop job --create product\_import --import --connect jdbc:mysql://localhost/retailer\_db --username root -P cloudera --table products --target-dir /user/cloudera/sqoop/product\_temp/ --incremental append --check-column product\_id --last-value 1345

"append" or "lastmodified"

storeing that record in meatastore for remembering for next operation.

execute job:

before inserting two more records for executing job:

sqoop job --exec product\_import

sqoop job --show product\_import

null => while imported as it as it is sqoop threated as String

null => '\\N'

assignments:

1 sqoop import with free form query using joins one two tables

2 incremental imports with sqoop jobs

3 sqoop imports using --direct, mulitpale mappers, when there is no primary key, usage of split-by and store results inotparquest.

4 incremental with lastmodified records

5 data types specified

6 habse import

7 hive import