SQOOP COMMANDS AND CONCEPTS WITH DESCRIPTION

# 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 uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.
* Sqoop is a bulk and batch data exchange tool that efficiently transport structured data from RDBMS to hadoop & vice versa using mappers.
* Below details applies to version Sqoop v1.4.6.

# Why Sqoop

Business benefits

* DB and DWH consolidation/migration
* Backup and availability
* Building of data lake
* Complex even processing (CEP)
* Cost effective

Technical benefits

* DB with out sqoop has lot of challenges
* Long process is avoided
* Complexity is avoided

# Architecture:

Sqoop client

Data base tables

Map

Map

Map

Record container class. A jar file

HDFS

HDFS

HDFS

Map reduce job

Import:

1. Sqoop contacts DB using JDBC to get meta data. (the column names and data types)
2. Then sqoop creates record container class. A jar file. This jar file will contain the DB to hadoop data type conversion, input split info and so on.
3. If the sqoop command uses more than one mapper, then again sqoop contacts DB using JDBC to get the boundary values.
4. Then with all the info, the sqoop job is submitted to YARN
5. Then Yarn architecture is carried on to import data.(if –direct is used, then database’s own utility will be used for transfer. Else JDBC is used)

Export:

1. Sqoop client contacts DB using JDBC to validate database,table mentioned in the command.
2. Record class containder jar file is created
3. Job is submitted to YARN.. Here boundary concept is not applied because of export.

# Installation of Sqoop

**Step** 1: Installing Sqoop

Untar the sqoop 1.4.6 tar package and move to the common directory and give the respective permissions.

cd ~/install

tar xvzf sqoop-1.4.6.bin hadoop-2.0.4-alpha.tar.gz

sudo mv sqoop-1.4.6.bin hadoop-2.0.4-alpha /usr/local/sqoop

(this /usr/local/ will be the common folder for all the softwares which we install here. Because, when ever other application looks for this application (for eg: when oozie looks for sqoop) there should be a default path to give all defaluts. So we choose /usr/local in our case)

**Step 2:** 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

(here $SQOOP\_HOME means /usr/local/sqoop . its defined in .bashrc as shown below.

“Export SQOOP\_HOME=/usr/local/sqoop”

It will be executed when the terminal is initiated.)

mv sqoop-env-template.sh sqoop-env.sh

echo 'export HADOOP\_COMMON\_HOME=/usr/local/hadoop' >> $SQOOP\_HOME/conf/sqoop-env.sh

echo 'export HADOOP\_MAPRED\_HOME=/usr/local/hadoop' >> $SQOOP\_HOME/conf/sqoop-env.sh

(above echo statements are, defining global variables in sqoop-env.. these variables are the path for hadoop libs.

When sqoop runs, it will search for hadoop libs. So the path is given in sqoop-env which will be initiated wen sqoop is initiated.)

**Step 3:** Copy mysql-connector-java

cp -p ~/install/mysql-connector-java.jar /usr/local/sqoop/

(in order to connect with DB, we need a DB connector. It’s a JDBC driver specific for each DB. Need to get the driver jar file from database team and need to place it inside Sqoop. Without this driver, connection to DB will not be done. )

**Step 4:** Verifying Sqoop

The following command is used to verify the Sqoop version.

sqoop-version

# Generic Syntax:

Sqoop ToolName (generic agruments) (tool specific arguments)

Sqoop-ToolName (generic agruments) (tool specific arguments)

* Note that generic Hadoop arguments are preceeded by a single dash character (-), whereas tool-specific arguments start with two dashes (--), unless they are single character arguments such

as -P.

# Using Option files:

* When using Sqoop, the command line options that do not change from invocation to invocation can be put in an options file for convenience
* Also supported are comments within option files that begin with the hash character, and it should be entered in a new line.

Eg:

Below command without option files:

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

Below common using option file:

* When the sqoop command is ran via Sqoop comman line, then the option file should be available in the local system where the sqoop client runs locally.

sqoop list-databases --option-file /user/local/dataset/connection.txt

* When the sqoop command is ran via OOZIE, then the option file should be available in the HDFS, because oozie submits a MR job and it runs in hadoop system.

sqoop list-databases --option-file /kannan/optionfile/connection.txt

# Sqoop-Import commands:

* The import tool imports an individual table from an RDBMS to HDFS.
* Each row from a table is represented as a separate record in HDFS.
* Records can be stored as text files (one record per line), or in binary representation as Avro or Sequence Files.

## Connecting to DB:

Sqoop --connect jdbc:mysql://localhost --username root--password root

The above command will fail because the tool is missing here. So we can use list-database or list-table or just simple import. Like shown below.

sqoop import --connect jdbc:mysql://localhost --username root--password root

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

*output:*

*information\_schema*

*classicmodels*

*mysql*

*testdb*

*vivadb*

sqoop list-tables --connect jdbc:mysql://localhost/vivadb --username root --password root

*output:*

*customer*

*employee*

*health\_care*

*sales\_rate*

*sales\_tran*

sqoop list-tables --connect jdbc:mysql://localhost/vivadb --username root -P

if the password part is mentioned as -P, then it will prompt the user to enter the password in command line while execution.

## Simple data import:

Key points to remember.

* Use the --table argument to select the table to import. For example, --table employees.

Argument can also identify a VIEW or other table-like entity in a database.

* By default all columns from a table are imported
* You can select a subset of columns and control their ordering by using the --columns argument. This should include a comma-delimited list of columns to import. For example: --columns "name,employee\_id,jobtitle".
* You can append a WHERE clause to this with the --where argument. For example: --where "id > 400"
* In some cases this query is not the most optimal so you can specify --boundary-query argument which will be discussed later.
* When we do simple import, then max/min values are fetched from table by sqoop directly.
* When we use free form query, sqoop executed the entire free form query as sub query and on top of it sqoop runs outer query to get boundary values. This is costly.
* In that case we can directly mention the boundary query in the command as the developer knows better than sqoop about the boundary queries.

One example:

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root --password root -table customers -m 3 --delete-target-dir ;

The table “customers” will be imported to the default home location of HDFS.

when the number of mappers are greater than 1, then we need to specify --split-by column or the table should contain any primary key column.. else it will throw the below error.

19/03/25 21:54:58 ERROR tool.ImportTool: Error during import: No primary key could be found for table customers. Please specify one with --split-by or perform a sequential import with '-m 1'.

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root --password root -table customers --split-by customerNumber -m 3 --delete-target-dir ;

when checking the log,

1. Point 1 in sqoop arch. sqoop contacts db using JDBC to get metadata

19/03/25 22:01:04 INFO manager.SqlManager: Executing SQL statement: SELECT t.\* FROM `customers` AS t LIMIT 1

19/03/25 22:01:04 INFO manager.SqlManager: Executing SQL statement: SELECT t.\* FROM `customers` AS t LIMIT 1

2. Point 2 in sqoop arch. sqoop creates record class container jar file

19/03/25 22:01:07 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hduser/compile/7b3df60f10ab8bf8a0555f4443550d0b/customers.jar

3. Point 3 in sqoop arch. If no of mappers are >1 then sqoop contacts DB using JDBC to get the bourdary values as shown below.

19/03/25 22:01:26 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`customerNumber`), MAX(`customerNumber`) FROM `customers`

4. Point 4 in sqoop arch. then Sqoop submits the jobs with all info to YARN.

19/03/25 22:01:27 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1553442825987\_0012

19/03/25 22:01:29 INFO impl.YarnClientImpl: Submitted application application\_1553442825987\_0012

## Free form query imports

* Instead of using the --table, --columns and --where arguments, you can specify a SQL statement with the --query argument.
* When importing a free-form query, you must specify a destination directory with --target-dir.
* Need to use $CONDITIONS in the query when we use free-form query imports.

Sqoop expects this keyword to replace this with an inbuilt condition to perform the import operation.

It helps Sqoop to do splits while importing

* When importing free-form query results in parallel, you must specify --split-by (even if table has primary key).

Eg:

*Select columns: (VERTICAL CONTROL)*

sqoop import --connect jdbc:mysql://localhost/vivadb --username root --password root --query 'SELECT eid,fname,lname,salary FROM employee WHERE $CONDITIONS' --target-dir '/kannan/freequery1/' --verbose --split-by eid

*select columns with WHERE clause (HORIZONTAL CONTROL)*

sqoop import --connect jdbc:mysql://localhost/vivadb --username root --password root --query 'SELECT eid,fname,lname,salary FROM employee WHERE eid > 2000 AND $CONDITIONS' --target-dir '/kannan/freequery1/' --verbose --split-by eid

*select columns with join using (USING)*

sqoop import --connect jdbc:mysql://localhost/vivadb --username root --password root --query 'SELECT employee.eid, employee.fname, employee.lname,department.did,department.deptname, salary FROM employee INNER JOIN department USING(did) WHERE $CONDITIONS' --target-dir '/kannan/freequery1/' --verbose --split-by eid

*select columns with join using (ON)*

sqoop import --connect jdbc:mysql://localhost/vivadb --username root --password root --query 'SELECT employee.eid, employee.fname, employee.lname,department.did,department.deptname, salary FROM employee INNER JOIN department on (employee.did == department.did) WHERE $CONDITIONS' --target-dir '/kannan/freequery1/' --verbose --split-by eid

|  |
| --- |
| Concepts:  Parallelism: *(-m , --num-mappers , --split-by and --autoreset-to-one-mapper)*   * No of mappers to be ran can be specified using -m or --num-mappers * If mapper not specified and if import table has primary key, then default is set to 4 mappers * If mapper not specified and if import table does not have primary key, then --split-by is mandatory. Else command will fail. * Split functionality - will split the import into equal parts and will run individual mapper task on each part in parallel. * If the import table does not have primary column and if split-by is not specified then the command can be executed successful by specifying the --autoreset-to-one-mapper command. But its usually used with import-all-tables. |

Example:

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root -P --query " select cus.customerNumber, cus.customerName, cus.phone, ord.orderNumber, ord.orderDate from customers cus inner join orders ord on cus.customerNumber=ord.customerNumber where (ord.orderDate between '2004-01-01' and '2004-01-15') and \$CONDITIONS " --target-dir filter\_cust --delete-target-dir -m 2 --split-by cus.customerNumber;

1. getting the meta data

19/03/25 22:40:13 INFO manager.SqlManager: Executing SQL statement: select cus.customerNumber, cus.customerName, cus.phone, ord.orderNumber, ord.orderDate from customers cus inner join orders ord on cus.customerNumber=ord.customerNumber where (ord.orderDate between '2004-01-01' and '2004-01-15') and (1 = 0)

2. creating the record class container jar file

19/03/25 22:45:16 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hduser/compile/d39a27209fa74f3187fae1b4946cff79/QueryResult.jar

3. getting boundary values

19/03/25 22:45:22 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(t1.customerNumber), MAX(t1.customerNumber) FROM ( select cus.customerNumber, cus.customerName, cus.phone, ord.orderNumber, ord.orderDate from customers cus inner join orders ord on cus.customerNumber=ord.customerNumber where (ord.orderDate between '2004-01-01' and '2004-01-15') and (1 = 1) ) AS t1

4.submitting job to YARN

19/03/25 22:45:22 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1553442825987\_0014

19/03/25 22:45:23 INFO impl.YarnClientImpl: Submitted application application\_1553442825987\_0014

* Now check the target dir

[hduser@Inceptez datanode]$ hadoop fs -ls filter\_cust

(or)

[hduser@Inceptez datanode]$ hadoop fs -ls /user/hduser/filter\_cust

Found 3 items

-rw-r--r-- 1 hduser hadoop 0 2019-03-25 22:45 /user/hduser/filter\_cust/\_SUCCESS

-rw-r--r-- 1 hduser hadoop 114 2019-03-25 22:45 /user/hduser/filter\_cust/part-m-00000

-rw-r--r-- 1 hduser hadoop 54 2019-03-25 22:45 /user/hduser/filter\_cust/part-m-00001

Note: \_SUCCESS is a zero byte file created by sqoop to indicate that the command has been executed successfully.

## --skip-dist-cache:

* Sqoop will copy the jars in $SQOOP\_HOME/lib folder to job cache every time when start a Sqoop job
* While sqoop is executed from OOZIE, it uses its own sqoop share libraries.
* So, sqoop copying from $SQOOP\_HOME/lib to cache is and extra task and it degrades the performance.
* To avoid it we can use the argument –skip-dist-cache. (but should be used only while using OOZIE)

Eg: *while running in command line,*

sqoop import --connect jdbc:mysql://localhost/vivadb --table employee --username root --password root --target-dir '/kannan/skipdist/' --validate --skip-dist-cache --verbose -m5

*output:*

/\* 18/05/10 08:18:28 ERROR tool.ImportTool: Error during import: Import job failed! \*/

## --target-dir VS --warehouse-dir:

|  |  |
| --- | --- |
| --target-dir | --warehouse-dir |
| * It imports the table to the specified target directory. * Usually used with import (single table) | * It creates a parent directory into which import is done. * Usually used with import-all-tables * It creates a parent directory and inside it sub directories are created for each table (inside it splits are created). |

Eg: --target-dir

sqoop import --connect jdbc:mysql://localhost/vivadb --table employee --username root --password root --target-dir '/kannan/skipdist/' --validate --verbose -m5

*output:*

*hduser@hadoop:/usr/local/hadoop-2.6.0/sbin$ hdfs dfs -ls /kannan/skipdist*

*Found 6 items*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 08:38 /kannan/skipdist/\_SUCCESS*

*-rw-r--r-- 1 hduser supergroup 1427 2018-05-10 08:38 /kannan/skipdist/part-m-00000*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 08:38 /kannan/skipdist/part-m-00001*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 08:38 /kannan/skipdist/part-m-00002*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 08:38 /kannan/skipdist/part-m-00003*

*-rw-r--r-- 1 hduser supergroup 442 2018-05-10 08:38 /kannan/skipdist/part-m-00004*

*Eg for –warehouse-dir -> we will see in import-all-tables.*

**--direct:**

* Some databases can perform imports in a more high-performance fashion by using database-specific data movement tools. For example, MySQL provides the mysqldump.
* By supplying the --direct argument, you are specifying that Sqoop should attempt the direct import channel. This channel may be higher performance than using JDBC.

Eg:

sqoop import --connect jdbc:mysql://localhost/vivadb --table employee --username root --password root --direct --default-character-set=latin1

*output:*

*18/05/10 09:14:04 INFO mapreduce.ImportJobBase: Transferred 1.8252 KB in 54.7021 seconds (34.1669 bytes/sec)*

*18/05/10 09:14:04 INFO mapreduce.ImportJobBase: Retrieved 42 records.*

*18/05/10 09:14:04 INFO util.AppendUtils: Appending to directory employee*

*hduser@hadoop:/usr/local/hadoop-2.6.0/sbin$ hdfs dfs -ls /user/hduser/employee*

*Found 4 items*

*-rw-r--r-- 1 hduser supergroup 1427 2018-05-10 09:13 /user/hduser/employee/part-m-00000*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 09:14 /user/hduser/employee/part-m-00001*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 09:14 /user/hduser/employee/part-m-00002*

*-rw-r--r-- 1 hduser supergroup 442 2018-05-10 09:14 /user/hduser/employee/part-m-00003*

* again running the same command

sqoop import --connect jdbc:mysql://localhost/vivadb --table employee --username root --password root --direct -- --default-character-set=latin1

*output:*

*18/05/10 09:10:01 ERROR tool.ImportTool: Encountered IOException running import job: org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory hdfs://localhost:50000/user/hduser/employee already exists*

*because the folder employee already exists in HDFS location before executing this query*

**--append:**

* To resolve the above error use --append

sqoop import --connect jdbc:mysql://localhost/vivadb --table employee --username root --password root --append -- --default-character-set=latin1

*output:*

*18/05/10 09:14:04 INFO mapreduce.ImportJobBase: Transferred 1.8252 KB in 54.7021 seconds (34.1669 bytes/sec)*

*18/05/10 09:14:04 INFO mapreduce.ImportJobBase: Retrieved 42 records.*

*18/05/10 09:14:04 INFO util.AppendUtils: Appending to directory employee*

*hduser@hadoop:/usr/local/hadoop-2.6.0/sbin$ hdfs dfs -ls /user/hduser/employee*

*Found 4 items*

*-rw-r--r-- 1 hduser supergroup 1427 2018-05-10 09:13 /user/hduser/employee/part-m-00000*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 09:14 /user/hduser/employee/part-m-00001*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 09:14 /user/hduser/employee/part-m-00002*

*-rw-r--r-- 1 hduser supergroup 442 2018-05-10 09:14 /user/hduser/employee/part-m-00003*

*-rw-r--r-- 1 hduser supergroup 1427 2018-05-10 09:16 /user/hduser/employee/part-m-00004*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 09:17 /user/hduser/employee/part-m-00005*

*-rw-r--r-- 1 hduser supergroup 0 2018-05-10 09:17 /user/hduser/employee/part-m-00006*

*-rw-r--r-- 1 hduser supergroup 442 2018-05-10 09:17 /user/hduser/employee/part-m-00007*

|  |
| --- |
| Concepts:  **Validate**: will validate whether the no of rows from selected from source and imported at the destination are in sync. As shown below.  *18/05/10 08:34:36 INFO mapreduce.ImportJobBase: Transferred 1.8252 KB in 80.4554 seconds (23.2303 bytes/sec)*  *18/05/10 08:34:36 INFO mapreduce.ImportJobBase: Retrieved 42 records.*  *18/05/10 08:34:36 DEBUG mapreduce.ImportJobBase: Validating imported data.*  *18/05/10 08:34:36 DEBUG manager.SqlManager: No connection paramenters specified. Using regular API for making connection.*  *18/05/10 08:34:36 INFO mapreduce.JobBase: Validating the integrity of the import using the following configuration*  *Validator : org.apache.sqoop.validation.RowCountValidator*  *Threshold Specifier : org.apache.sqoop.validation.AbsoluteValidationThreshold*  *Failure Handler : org.apache.sqoop.validation.AbortOnFailureHandler*  *18/05/10 08:34:36 DEBUG validation.RowCountValidator: Validating data using row counts: Source [42] with Target[42]*  *18/05/10 08:34:36 DEBUG validation.AbsoluteValidationThreshold: Absolute Validation threshold comparing 42 with 42*  *18/05/10 08:34:36 INFO validation.RowCountValidator: Data successfully validated*  *18/05/10 08:34:36 DEBUG util.ClassLoaderStack: Restoring classloader:* [*sun.misc.Launcher$AppClassLoader@591ce4fe*](mailto:sun.misc.Launcher$AppClassLoader@591ce4fe)  **Verbose**: Print more information in console while working  **--relaxed-isolation** : reads uncommitted rows from source . |

**Fetch size:**

* Using fetch size command we can control the no of rows getting fetched into sqoop memory while doing imports.
* The default fetch size is 1000 rows
* By defining fetch size, in eg: only 100 rows will be loaded into sqoop memory and will be process before the next set of 100 rows are fetched from DB and loaded into sqoop memory.

Eg:

Scoop import --connect jdbc:mysql://localhost/classicmodels --username root –P --table customers --m 3 --split-by customerNumber --fetch-size 100.

**Fields terminated by & lines terminated by:**

* When a table is imported to HDFS, the columns in the table will be delimited by any special character.
* By default in Sqoop import the delimiter is comma ( , )
* Similarly each row will be import as a line. Each line will be terminated by ( \n ) as default
* It can be overridden by specifying corresponding commands in the sqoop statement.

Eg:

Sqoop import –connect jdbc:mysql://localhost/classicmodels --username root –P --table customers --m 3 --split-by customerNumber --fields-terminated-by ‘~’ --lines-terminated-by ‘\n’

**Sqoop Jobs:**

* A sqoop job creates and saved sqoop import & export commands.
* Can be executed multiple times.. usually used in incremental imports.
* The sqoop job will be saved in the Sqoop meta store

Syntax:

Sqoop job --create job\_name -- import ………….remaining all same as usual import statement

Eg: create

sqoop job --create myjob -- import --connect jdbc:mysql://localhost/vivadb --table employee --username root --password root --target-dir '/outputdir/job1' --validate --verbose -m5

Eg: List the jobs created

sqoop job --list

eg: To show parameters of a job

sqoop job --show myjob

eg: To execute a job

sqoop job --exec myjob

eg: Delete a job

sqoop job --delete myincjob

|  |
| --- |
| Concepts:   * Composite primary keys will not work in sqoop * Once the import is done, sqoop generates a \_SUCCESS file as a flag of successful completion. In real time the following command will be added in script to validate the successful completion.   🡪 if hadoop fs –ls /user/hduser/customers/\_SUCCESS | wc -l == 1 |

**Incremental Imports:**

* Sqoop provides an incremental import mode which can be used to retrieve only rows newer than some previously-imported set of rows.
* There are two types:
* Append (Append mode are used when the check column is kind of a row id that will have increasing values from the previous imports.)
* Last modified (Specifies the last value of the check column from previous import.)

**Eg: Append**

Do an initial import

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root -P --table customers -m 3 --split-by customerNumber

check the HDFS imported location.

hadoop fs -ls /user/hduser/customers

Found 4 items

-rw-r--r-- 1 hduser hadoop 0 2019-03-26 00:27 /user/hduser/customers/\_SUCCESS

-rw-r--r-- 1 hduser hadoop 5209 2019-03-26 00:27 /user/hduser/customers/part-m-00000

-rw-r--r-- 1 hduser hadoop 4646 2019-03-26 00:27 /user/hduser/customers/part-m-00001

-rw-r--r-- 1 hduser hadoop 3981 2019-03-26 00:27 /user/hduser/customers/part-m-00002

now insert couple of rows in the DB. And import again with last value given manually by developer.

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root -P --table customers -m 3 --split-by customerNumber --incremental append --check-column customerNumber --last-value 480

now look at the imported directory: part-m-files 3 to 5 are created.

hadoop fs -ls /user/hduser/customers

Found 7 items

-rw-r--r-- 1 hduser hadoop 0 2019-03-26 00:27 /user/hduser/customers/\_SUCCESS

-rw-r--r-- 1 hduser hadoop 5209 2019-03-26 00:27 /user/hduser/customers/part-m-00000

-rw-r--r-- 1 hduser hadoop 4646 2019-03-26 00:27 /user/hduser/customers/part-m-00001

-rw-r--r-- 1 hduser hadoop 3981 2019-03-26 00:27 /user/hduser/customers/part-m-00002

-rw-r--r-- 1 hduser hadoop 695 2019-03-26 00:45 /user/hduser/customers/part-m-00003

-rw-r--r-- 1 hduser hadoop 0 2019-03-26 00:45 /user/hduser/customers/part-m-00004

-rw-r--r-- 1 hduser hadoop 184 2019-03-26 00:45 /user/hduser/customers/part-m-00005

check the log of previous import: boundary values are set as per our manually entered value and max value.

19/03/26 00:44:50 INFO tool.ImportTool: Incremental import based on column `customerNumber`

19/03/26 00:44:50 INFO tool.ImportTool: Lower bound value: 480

19/03/26 00:44:50 INFO tool.ImportTool: Upper bound value: 1751

19/03/26 00:44:50 WARN manager.MySQLManager: It looks like you are importing from mysql.

19/03/26 00:44:50 WARN manager.MySQLManager: This transfer can be faster! Use the --direct

19/03/26 00:44:50 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.

After getting the boundary value,

sqoop recognized that’s its incremental import and is getting the lower bound value from the value which we specified in --last-value.

and gets the upper bound value from the boundary query sqoop executed just now.

Also Sqoop recognizes that we are importing from MYSQL DB and it suggests to use --direct parameter to make the import faster.

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root -P --table customers -m 3 --split-by customerNumber --incremental append --check-column city --last-value 'chennai'

As shown above if we use a non-numeric in --last-value, then sqoop will throw an error.

19/03/26 00:52:17 ERROR tool.ImportTool: Error during import: Character column (city) cannot be used to determine which rows to incrementally import.

In real time we cannot remember the --last-value after each import. Sqoop should remember it. for that we should create incremental import statements as a job.

so that the job which will be save in the metastore will save the last value as well along with it.

sqoop job --create myjob1 -- import --connect jdbc:mysql://localhost/classicmodels --username root -P --table customers -m 3 --split-by customerNumber --target-dir savedjob1 --incremental append --check-column customerNumber --last-value 0

sqoop job --exec myjob1

look at the log:

boundry will be considering the entire table as the --last-value is given as 0..

first it takes min,max from table

then forms boundry values using the min/max.

19/03/26 01:02:57 INFO tool.ImportTool: Maximal id query for free form incremental import: SELECT MAX(`customerNumber`) FROM `customers`

19/03/26 01:02:57 INFO tool.ImportTool: Incremental import based on column `customerNumber`

19/03/26 01:02:57 INFO tool.ImportTool: Lower bound value: 0

19/03/26 01:02:57 INFO tool.ImportTool: Upper bound value: 495

....................

....................

19/03/26 01:03:02 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`customerNumber`), MAX(`customerNumber`) FROM `customers` WHERE ( `customerNumber` > 0 AND `customerNumber` <= 495 )

look at the target dir after initial import

hadoop fs -ls /user/hduser/savedjob1

Found 3 items

-rw-r--r-- 1 hduser hadoop 5209 2019-03-26 01:03 /user/hduser/savedjob1/part-m-00000

-rw-r--r-- 1 hduser hadoop 4646 2019-03-26 01:03 /user/hduser/savedjob1/part-m-00001

-rw-r--r-- 1 hduser hadoop 3981 2019-03-26 01:03 /user/hduser/savedjob1/part-m-00002

now i insert couple of new rows.

and i execute the sqoop job myjob1 again.

sqoop job --exec myjob1

last value was taken from the sqoop job meta data from metastore.

19/03/26 01:10:13 INFO tool.ImportTool: Maximal id query for free form incremental import: SELECT MAX(`customerNumber`) FROM `customers`

19/03/26 01:10:13 INFO tool.ImportTool: Incremental import based on column `customerNumber`

19/03/26 01:10:13 INFO tool.ImportTool: Lower bound value: 495

19/03/26 01:10:13 INFO tool.ImportTool: Upper bound value: 1751

............

...........

19/03/26 01:10:18 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`customerNumber`), MAX(`customerNumber`) FROM `customers` WHERE ( `customerNumber` > 495 AND `customerNumber` <= 1751 )

check the target dir:

hadoop fs -ls /user/hduser/savedjob1

Found 5 items

-rw-r--r-- 1 hduser hadoop 5209 2019-03-26 01:03 /user/hduser/savedjob1/part-m-00000

-rw-r--r-- 1 hduser hadoop 4646 2019-03-26 01:03 /user/hduser/savedjob1/part-m-00001

-rw-r--r-- 1 hduser hadoop 3981 2019-03-26 01:03 /user/hduser/savedjob1/part-m-00002

-rw-r--r-- 1 hduser hadoop 90 2019-03-26 01:10 /user/hduser/savedjob1/part-m-00003

-rw-r--r-- 1 hduser hadoop 94 2019-03-26 01:10 /user/hduser/savedjob1/part-m-00004

|  |
| --- |
| Concepts:   * Metadata in sqoop such as job info, last values are stored in hsql data base. * Hsql is hyper SQL.. this is the default database for sqoop meta data |

**EG: Last modified:**

sqoop job --create pay -- import --connect jdbc:mysql://localhost/classicmodels --username root -P --table payments -m 3 --split-by customerNumber --target-dir '/user/hduser/sqoop/imports/payments' --incremental lastmodified --check-column paymentDate --last-value 0

Execute the job:

sqoop job --exec pay

check the log:

Note the lower and upper bound values

19/04/04 15:08:27 INFO manager.SqlManager: Executing SQL statement: SELECT t.\* FROM `payments` AS t LIMIT 1

19/04/04 15:08:27 INFO tool.ImportTool: Incremental import based on column `paymentDate`

19/04/04 15:08:27 INFO tool.ImportTool: Lower bound value: '0'

19/04/04 15:08:27 INFO tool.ImportTool: Upper bound value: '2019-04-04 15:08:27.0'

..........

19/04/04 15:09:16 INFO mapreduce.ImportJobBase: Retrieved 20 records.

19/04/04 15:09:16 INFO tool.ImportTool: Saving incremental import state to the metastore

19/04/04 15:09:16 INFO tool.ImportTool: Updated data for job: pay

Check the target dir:

hadoop fs -ls /user/hduser/sqoop/imports/payments

Found 4 items

-rw-r--r-- 1 hduser hadoop 0 2019-04-04 14:44 /user/hduser/sqoop/imports/payments/\_SUCCESS

-rw-r--r-- 1 hduser hadoop 97 2019-04-04 14:44 /user/hduser/sqoop/imports/payments/part-m-00000

-rw-r--r-- 1 hduser hadoop 227 2019-04-04 14:44 /user/hduser/sqoop/imports/payments/part-m-00001

-rw-r--r-- 1 hduser hadoop 329 2019-04-04 14:44 /user/hduser/sqoop/imports/payments/part-m-00002

hadoop fs -cat /user/hduser/sqoop/imports/payments/\*

101,HQ336336,2004-10-19,6066.78

102,JM555205,2003-06-05,14571.44

103,OM314933,2004-12-18,1676.14

104,BO864823,2004-12-17,14191.12

105,HQ55022,2003-06-06,32641.98

106,ND748579,2004-08-20,33347.88

107,GG31455,2003-05-20,45864.03

108,MA765515,2004-12-15,82261.22

109,NP603840,2003-05-31,7565.08

110,NR27552,2004-03-10,44894.74

111,DB933704,2004-11-14,19501.82

112,LN373447,2004-08-08,47924.19

113,NG94694,2005-02-22,49523.67

114,DB889831,2003-02-16,50218.95

115,FD317790,2003-10-28,1491.38

116,KI831359,2004-11-04,17876.32

117,MA302151,2004-11-28,34638.14

118,AE215433,2005-03-05,101244.59

119,BG255406,2004-08-28,85410.87

120,CQ287967,2003-04-11,11044.30

run the job again

sqoop job --exec pay

now while running the job for the second time the job it fails. Because we need to mention what we need to do if existing data is modified

19/04/04 15:13:36 ERROR tool.ImportTool: Error during import: --merge-key or --append is required when using --incremental lastmodified and the output directory exists.

Either we need to modify it in HDFS. or append the modified row as new data.

--merge-key (it will modify if any changes in existing/will append new data and will merge all available files into a single file)

or

--append (will append as new data. Not matter if its new data or changes to existing data. In this case duplicates will be created)

Now lets delete the job pay and recreate it with --merge-key:

sqoop job --create pay -- import --connect jdbc:mysql://localhost/classicmodels --username root -P --table payments -m 3 --split-by customerNumber --target-dir '/user/hduser/sqoop/imports/payments' --incremental lastmodified --check-column paymentDate --last-value 0 --merge-key customerNumber

clean up the target dir and do the first import again as we did above.

after first import is done,

Note the upper bound value is **'2019-04-04 15:08:27.0'** as we can see in the above log, the --last-value in the metastore will be saved as same '2019-04-04 15:08:27.0' which is current timestamp.

so now to practice lets modify it to some past date as show below.

-go to home by cd ~

-give ls -lart to find the hidden folder .sqoop

-enter .sqoop to see the file metastore.db.script.

-open it by vi metastore.db.script..

-change the lastvalue to old date as shown below.

INSERT INTO SQOOP\_SESSIONS VALUES('pay','incremental.last.value','2010-04-04 15:58:56.0','SqoopOptions')

now insert some new rows with paymentDate as recent past like 2017/2018.

and update some existing rows as shown below.

update payments set amount = 100, paymentDate = '2017-08-18' where customerNumber in (112,113,114,115) ;

run job pay again

Note the log:

19/04/04 22:13:53 INFO manager.SqlManager: Executing SQL statement: SELECT t.\* FROM `payments` AS t LIMIT 1

19/04/04 22:13:53 INFO tool.ImportTool: Incremental import based on column `paymentDate`

19/04/04 22:13:53 INFO tool.ImportTool: Lower bound value: '2010-04-05 07:34:15.0'

19/04/04 22:13:53 INFO tool.ImportTool: Upper bound value: '2019-04-05 07:43:53.0'

101,HQ336336,2004-10-19,6066.78

102,JM555205,2003-06-05,14571.44

103,OM314933,2004-12-18,1676.14

104,BO864823,2004-12-17,14191.12

105,HQ55022,2003-06-06,32641.98

106,ND748579,2004-08-20,33347.88

107,GG31455,2003-05-20,45864.03

108,MA765515,2004-12-15,82261.22

109,NP603840,2003-05-31,7565.08

110,NR27552,2004-03-10,44894.74

111,DB933704,2004-11-14,19501.82

112,LN373447,2017-08-18,100.00

113,NG94694,2017-08-18,100.00

114,DB889831,2017-08-18,100.00

115,FD317790,2017-08-18,100.00

116,KI831359,2004-11-04,17876.32

117,MA302151,2004-11-28,34638.14

118,AE215433,2005-03-05,101244.59

119,BG255406,2004-08-28,85410.87

120,CQ287967,2003-04-11,11044.30

121,ET64396,2017-04-16,83598.04

122,HI366474,2017-04-27,47142.70

123,HR86578,2017-04-12,55639.66

124,KI131716,2018-04-15,111654.40

125,LF217299,2018-04-26,43369.30

126,NT141748,2018-04-25,45084.38

127,DI925118,2018-04-28,10549.01

128,FA465482,2018-04-18,24101.81

129,FH668230,2018-04-24,33820.62

130,IP383901,2018-04-18,7466.32

***NOTE****:*

1. *If we use last modified, then the –check column should be a date or a timestamp.*
2. *We have given “*--last-value '2018-05-20 09:34:10'” . so the rows which have timestamp before this value will not be picked up in the import.

**--boundary-query:**

* As we have discussed before, sqoop takes care of getting the boundary query when we use –split-by column.
* If we use a complex free form query, then the query will be executed to get the boundary values which consumes lot of CPU and time.
* To avoid that, since developer know well about the boundary values, he can explicitly give the simplified query to get the boundary values.
* Boundry-query is part of performance tuning. It gives faster retrieval.
* Run this , this without boundary query

sqoop import --connect jdbc:mysql://localhost/classicmodels --username root -P --query "select cus.customerNumber,pay.paymentDate,pay.amount from customers cus inner join payments pay on cus.customerNumber = pay.customerNumber where (pay.paymentDate between '2002-01-01' and '2005-12-31') and \$CONDITIONS" --boundary-query 'select min(customerNumber) , max(customerNumber) from customers' --split-by cus.customerNumber -m 3 --target-dir '/user/hduser/sqoop/imports/boundary\_qry'

check the log:

The entire query is executed along with (1=0) to get metadata.

If we don’t use –boundary-query again the entire query will run again along with the condition (1=1) which consumes time and CPU.

19/04/05 02:48:53 INFO manager.SqlManager: Executing SQL statement: select cus.customerNumber,pay.paymentDate,pay.amount from customers cus inner join payments pay on cus.customerNumber = pay.customerNumber where (pay.paymentDate between '2002-01-01' and '2005-12-31') and (1 = 0)

Note the boundary query which was mentioned in the sqoop command is used here to get the boundary values.

19/04/05 02:50:07 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: select min(customerNumber) , max(customerNumber) from customers

19/04/05 02:50:07 INFO mapreduce.JobSubmitter: number of splits:3

Check the target dir:

hadoop fs -cat /user/hduser/sqoop/imports/boundary\_qry/\*

103,2004-12-18,1676.14

119,2004-08-28,85410.87

**Enable Compression:**

* Without compression-

sqoop-import --connect jdbc:mysql://localhost/vivadb --username root --password root --table employee --delete-target-dir --target-dir /inputdir/input/sqoop/empwithoutcomp –verbose

*Output:*

*File Output Format Counters*

*Bytes Written=1869*

*18/05/14 09:35:42 INFO mapreduce.ImportJobBase: Transferred 1.8252 KB in 49.5451 seconds (37.7232 bytes/sec)*

*18/05/14 09:35:42 INFO mapreduce.ImportJobBase: Retrieved 42 records.*

* With Compression you can see the storage in HDFS is reduced by 40%
* By default its GZIP compression format.

sqoop-import --connect jdbc:mysql://localhost/vivadb --username root --password root --table employee --delete-target-dir --target-dir /inputdir/input/sqoop/empseq --verbose -z –compress

*File Output Format Counters*

*Bytes Written=1033*

*18/05/14 09:33:11 INFO mapreduce.ImportJobBase: Transferred 1.0088 KB in 57.192 seconds (18.062 bytes/sec)*

*18/05/14 09:33:11 INFO mapreduce.ImportJobBase: Retrieved 42 records.*

* --compression-codec <c> : You can also supplement the compression codec that you want use explicitly using this command.

sqoop-import --connect jdbc:mysql://localhost/vivadb --username root --password root --table employee --delete-target-dir --target-dir /inputdir/input/sqoop/empseq --verbose --compression-codec snappy

# Sqoop-Import-All commands:

A tool which imports a set of tables from an RDBMS to HDFS is what we call the import-all-tables. Basically, here in HDFS, data from each table is stored in a separate directory.

However, there are various conditions that must meet for the import-all-tables tool to be useful. Such as:

1. Basically, it is important that each table must have a single-column primary key. Also, –autoreset-to-one-mapper option must be used.
2. Must intend to import all columns of each table.
3. Must not intend to use non-default splitting column, nor impose any conditions via WHERE clause

if you do not have primary key you cannot use split by the way we have done in simple import instead you can use **auto reset** to one mapper and for all those tables where there are no primary key it will use only one thread to copy the data.

We can import either the set of tables which has primary keys with multiple mappers, or set of tables which doesn’t have primary keys with auto reset to one mapper option.

Combining tables with and without primary keys will not work as expected unless we use auto reset to one mapper. If we use this option, then split will not happen while importing.

**import-all-tables with target dir:**

sqoop import-all-tables --connect jdbc:mysql://localhost/classicmodels --username root -P --target-dir '/user/hduser/sqoop/imports/classicmodels\_full'

the above command will abend because, we have used –target-dir.

For import-all we need to use –warehouse-dir

sqoop import-all-tables --connect jdbc:mysql://localhost/classicmodels --username root -P --warehouse-dir '/user/hduser/sqoop/imports/classicmodels\_full'

the above command failed again because it dint have primary columns.

* If all the tables in the imports statement has primary keys, then sqoops default split count will be considered. Or we can specify the mappers and split-by column.
* If no primary columns, then need to make it one mapper (-m 1)

else -–autoreset-to-one-mapper

19/04/05 03:40:39 ERROR tool.ImportAllTablesTool: Error during import: No primary key could be found for table customers. Please specify one with --split-by or perform a sequential import with '-m 1'.

Now let’s run the modified command:

sqoop import-all-tables --connect jdbc:mysql://localhost/classicmodels --username root -P --warehouse-dir '/user/hduser/sqoop/imports/classicmodels\_full' --autoreset-to-one-mapper

(or)

sqoop import-all-tables --connect jdbc:mysql://localhost/classicmodels --username root -P --warehouse-dir '/user/hduser/sqoop/imports/classicmodels\_full' -m 1

check the log:

19/04/05 03:45:44 WARN manager.SqlManager: Split by column not provided or can't be inferred. Resetting to one mapper

...............

19/04/05 03:44:40 INFO mapreduce.JobSubmitter: number of splits:1

Let’s see the warehouse dir:

Each table is imported as a directory under the warehouse dir calssicmodels\_full.

hadoop fs -ls /user/hduser/sqoop/imports/classicmodels\_full

Found 3 items

drwxr-xr-x - hduser hadoop 0 2019-04-05 03:45 /user/hduser/sqoop/imports/classicmodels\_full/customers

drwxr-xr-x - hduser hadoop 0 2019-04-05 03:45 /user/hduser/sqoop/imports/classicmodels\_full/orders

drwxr-xr-x - hduser hadoop 0 2019-04-05 03:46 /user/hduser/sqoop/imports/classicmodels\_full/payments

further go ahead and open customers to see the imported file:

hadoop fs -ls /user/hduser/sqoop/imports/classicmodels\_full/customers

one file found for the table customers:

Found 2 items

-rw-r--r-- 1 hduser hadoop 0 2019-04-05 03:45 /user/hduser/sqoop/imports/classicmodels\_full/customers/\_SUCCESS

-rw-r--r-- 1 hduser hadoop 14020 2019-04-05 03:45 /user/hduser/sqoop/imports/classicmodels\_full/customers/part-m-00000

**import-all-tables with exclude tables:**

sqoop import-all-tables --connect jdbc:mysql://localhost/classicmodels --username root -P --warehouse-dir '/user/hduser/sqoop/imports/classicmodels\_full' -m 1 --exclude-tables customers,orders

* Note: only mentioned tables were imported.

**import-all-tables with include tables:**

sqoop import-all-tables --connect jdbc:mysql://localhost/classicmodels --username root -P --warehouse-dir '/user/hduser/sqoop/imports/classicmodels\_full' -m 1 --include-tables customers,orders

* Note: only mentioned tables were imported.

# Sqoop Export:

* The export tool exports a set of files from HDFS back to an RDBMS.
* The target table must already exist in the database
* The input files are read and parsed into a set of records according to the user-specified delimiters.
* Explanation about the below 3 modes
* The default operation is to transform these into a set of **INSERT** statements that inject the records into the database.
* In "update mode," Sqoop will generate **UPDATE** statements that replace existing records in the database.
* In "call mode" Sqoop will make a **stored procedure** call for each record.

**INSERT (the default mode of export):**

Lets create a file in HDFS to import into the mysql table payments.

hadoop fs -cat /user/hduser/sqoop/imports/payments/pay\_imp

131~HQ336336~2004-10-19~6066.78

132~JM555205~2003-06-05~14571.44

133~OM314933~2004-12-18~1676.14

134~BO864823~2004-12-17~14191.12

135~HQ55022~2003-06-06~32641.98

136~ND748579~2004-08-20~33347.88

137~FA465482~2018-04-18~24101.81

138~FH668230~2018-04-24~33820.62

139~IP383901~2018-04-18~7466.32

sqoop export -connect jdbc:mysql://localhost/classicmodels --username root -P --table payments --export-dir /user/hduser/sqoop/imports/payments/pay\_imp

the above command fails. Because the data in file is delimited by ‘~’. But since we dint mention any delimiting argument in command, sqoop takes its default arguments

|  |
| --- |
| Concepts:   * ‘,’ comma is default delimiter. ‘\n’ is the default line delimiter. |

19/04/05 09:02:26 INFO mapreduce.Job: Task Id : attempt\_1554348367183\_0031\_m\_000001\_2, Status : FAILED

Error: java.io.IOException: Can't export data, please check failed map task logs

.............

Caused by: java.lang.RuntimeException: Can't parse input data: '131~HQ336336~2004-10-19~6066.78'

................

Caused by: java.lang.NumberFormatException: For input string: "131~HQ336336~2004-10-19~6066.78"

at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)

...............

19/04/05 09:02:42 ERROR tool.ExportTool: Error during export: Export job failed!

Now modify and run the command with delimiting details:

sqoop export -connect jdbc:mysql://localhost/classicmodels \

--username root -P \

--table payments --export-dir /user/hduser/sqoop/imports/payments/pay\_imp \

--fields-terminated-by '~' \

--lines-terminated-by '\n'

now check the table:

select \* from payments;

The source data is imported into the table.

..........

..........

| 131 | HQ336336 | 2004-10-19 | 6066.78 |

| 132 | JM555205 | 2003-06-05 | 14571.44 |

| 133 | OM314933 | 2004-12-18 | 1676.14 |

| 138 | FH668230 | 2018-04-24 | 33820.62 |

| 139 | IP383901 | 2018-04-18 | 7466.32 |

| 136 | ND748579 | 2004-08-20 | 33347.88 |

| 137 | FA465482 | 2018-04-18 | 24101.81 |

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

**Update mode:**

There are two modes in Update.

1. **Updateonly (will only update any modification in existing data. New records from source will be skipped)**
2. **Allowinsert (will both update and insert. Will update the changes in existing data and will insert new data)**

**Now lets see the updateonly mode:**

now i have udpate some values in the existing source file and have inserted one new record.

131~HQ336336~2004-10-19~6066.78

132~JM555205~2003-06-05~10000.44

133~OM314933~2004-12-18~1676.14

134~BO864823~2004-12-17~10000.12

135~HQ55022~2003-06-06~32641.98

136~ND748579~2004-08-20~10000.88

137~FA465482~2018-04-18~24101.81

138~FH668230~2018-04-24~33820.62

139~IP223233~2018-04-18~100000.32

140~KJ099876~2019-04-04~444444

Run the sqoop command with update only.

sqoop export -connect jdbc:mysql://localhost/classicmodels \

--username root -P \

--table payments --export-dir /user/hduser/sqoop/imports/payments/pay\_imp \

--fields-terminated-by '~' \

--lines-terminated-by '\n' \

--update-key customerNumber \

--update-mode updateonly

Check the table.

The modified values in the file has been updated in the table.

NOTE: the new record in the file has not been inserted. Since its **updateonly**.

……

………………

| 130 | IP383901 | 2018-04-18 | 7466.32 |

| 131 | HQ336336 | 2004-10-19 | 6066.78 |

| 132 | JM555205 | 2003-06-05 | 10000.44 |

| 133 | OM314933 | 2004-12-18 | 1676.14 |

| 134 | BO864823 | 2004-12-17 | 10000.12 |

| 135 | HQ55022 | 2003-06-06 | 32641.98 |

| 136 | ND748579 | 2004-08-20 | 10000.88 |

| 137 | FA465482 | 2018-04-18 | 24101.81 |

| 138 | FH668230 | 2018-04-24 | 33820.62 |

| 139 | IP223233 | 2018-04-18 | 100000.32 |

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

|  |
| --- |
| Concepts:   * 19/04/05 09:21:56 INFO mapreduce.Job: Job job\_1554348367183\_0033 running in uber mode : false * **UBER mode** in YARN Hadoop2 - Running MapReduce jobs in small dataset. In normally mappers and reducers will run by ResourceManager (RM), RM will create separate container for mapper and reducer. **uber** configuration, will allow to run mapper and reducers in the same process as the ApplicationMaster (AM) |

**Now lets see the allowinsert mode:**

Now lets modify some existing data and will insert new records into the source file.

131~HQ336336~2004-10-19~6066.78

132~JM555205~2003-06-05~10000.44

133~OUCHANGE~2004-12-18~1676.14

134~BOCHANGE~2004-12-17~10000.12

135~HQ55022~2003-06-06~32641.98

136~ND748579~2004-08-20~10000.88

137~FACHANGE~2018-04-18~24101.81

138~FH668230~2018-04-24~33820.62

139~IPCHANGE~2018-04-18~100000.32

140~KJ099NEW~2019-04-04~444444

141~JH898NEW~2019-03-04~923723.32

142~KJ988NEW~2019-02-03~927739.2

143~KJ232NEW~2019-02-02~89768.2

144~JK233NEW~2019-03-04~98723.3

145~JK232NEW~2019-02-04~90873.4

sqoop export -connect jdbc:mysql://localhost/classicmodels \

--username root -P \

--table payments --export-dir /user/hduser/sqoop/imports/payments/pay\_imp \

--fields-terminated-by '~' \

--lines-terminated-by '\n' \

--update-key customerNumber \

--update-mode allowinsert

| 132 | JM555205 | 2003-06-05 | 10000.44 |

| 133 | OUCHANGE | 2004-12-18 | 1676.14 |

| 133 | OM314933 | 2004-12-18 | 1676.14 |

| 134 | BOCHANGE | 2004-12-17 | 10000.12 |

| 134 | BO864823 | 2004-12-17 | 10000.12 |

| 135 | HQ55022 | 2003-06-06 | 32641.98 |

| 135 | HQ55022 | 2003-06-06 | 32641.98 |

| 136 | ND748579 | 2004-08-20 | 10000.88 |

| 136 | ND748579 | 2004-08-20 | 10000.88 |

| 137 | FA465482 | 2018-04-18 | 24101.81 |

| 137 | FACHANGE | 2018-04-18 | 24101.81 |

| 138 | FH668230 | 2018-04-24 | 33820.62 |

| 138 | FH668230 | 2018-04-24 | 33820.62 |

| 139 | IP223233 | 2018-04-18 | 100000.32 |

| 139 | IPCHANGE | 2018-04-18 | 100000.32 |

| 140 | KJ099NEW | 2019-04-04 | 444444.00 |

| 141 | JH898NEW | 2019-03-04 | 923723.32 |

| 142 | KJ988NEW | 2019-02-03 | 927739.20 |

| 143 | KJ232NEW | 2019-02-02 | 89768.20 |

| 144 | JK233NEW | 2019-03-04 | 98723.30 |

| 145 | JK232NEW | 2019-02-04 | 90873.40 |

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

* --call (call a stored procedure using the data extracted by the export command)

DELIMITER //

CREATE PROCEDURE employee\_detail

(IN empid int(11))

BEGIN

SELECT name, deg FROM employee

WHERE id = empid;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE employee\_del

(IN empid int(11))

BEGIN

delete FROM employee

WHERE id = empid;

END //

DELIMITER ;

sqoop export --connect jdbc:mysql://localhost/sqoop --username root --password root --call employee\_detail --export-dir /inputdir/emp\_inp

*output:*

*(the file emp\_imp is having a list of ids which is available in the table exployee)*

*By executing the above command, the list of ids from file is extracted and passed as input to the mentioned stored procedure (and the select statement in the SP is executed)*

*Another example:*

sqoop export --connect jdbc:mysql://localhost/sqoop --username root --password root --call employee\_del --export-dir /kannan/sqoopexp/Scoop/emp\_inp

# sqoop-eval

* The eval tool allows users to quickly run simple SQL queries against a database; results are printed to the console.
* This allows users to preview their import queries to ensure they import the data they expect.

Eg;

sqoop eval --connect jdbc:mysql://localhost/vivadb --username root --password root --query 'SELECT eid,fname,lname,salary FROM employee LIMIT 10’

# sqoop-codegen

* The codegen tool generates Java classes which encapsulate and interpret imported records.
* The Java definition of a record is instantiated as part of the import process, but can also be performed separately.
* For example, if Java source is lost, it can be recreated. New versions of a class can be created which use different delimiters between fields, and so on.

# Dealing with Staging table while export:

**(It’s is to avoid partial insert into the DB)**

* When an export is done, the records in the HDFS is inserted into DB row by row.
* So if in the middle of process if someone queries the DB table, there is a possibility of fetching partial data.
* And if the export process fails, then the partial data will be inserted and others will be failed.
* In order to avoid this situation we use a intermediate table called staging table.
* The structure of staging table will be identical as the actual table.
* When we use staging table, the records will be inserted into staging row by row and when the last record is inserted, the who data from staging table will be inserted into the actual table in one shot.
* The clear-staging-table argument will clear the staging table once the export process is done.

sqoop export -Dsqoop.export.records.per.statement=5 \

--connect jdbc:mysql://localhost/custdb --username root --password root \

--table customer\_exp --export-dir cust\_exp \

--batch \

--staging-table customer\_stage \

--clear-staging-table \

--columns custid,fullname,age

|  |
| --- |
| Concepts:   * By using --columns in export, we can export the data into specific columns in DB table |

# Batch export:

By default if an export command is issued, sqoop opens the connection to DB and terminated the connection for each and every record. Because each and every record is converted as insert statement by sqoop and issued into DB.

To avoid such multiple connections/disconnections we can set the export-transaction properties count by below command.

By the below command, sqoop initiates connection and terminates it for every set of 10 records.

Note: if we give huge count like 10k, such huge no of records will sit in DBs memory before getting committed. It will become a burden to DB. So we need to choose the number wisely.

Sqoop export -Dsqoop.export.statements.per.transaction=10 --connect jdbc:mysql://localhost/classicmodels --username root -P --table paymnets --export-dir /kannan/sqoopexp/Scoop/emp\_inp

# Dealing with Null:

While importing data from DB to HDFS, we need to decide what value should be imported when the column in DB is having NULL.

It can be achieved by the arguments --null-non-string ‘0’ and --null-string ‘N/A’

sqoop import --connect jdbc:mysql://localhost/custdb --username root -P --boundary-query "select min(custid), max(custid) from customers" --query 'Select a.custid master\_custid,a.firstname,a.age,a.city,b.custid detail\_custid,a.createdt,b.fulladdress,category,transactiondt,transactamt from customers a join customer\_details b on a.custid=b.custid WHERE $CONDITIONS' --split-by a.custid --target-dir cust\_details --null- non-string '0' --null-string 'NA' --compress --direct --num-mappers 3 --fetch-size 100 --delete-target-dir;

# Exporting multiple tables using stored procedure

1. **Create file and type data into it**

vi empinfo 101,raja,dept1,Accounts 102,vinay,dept2,Finanace 103,karthik,dept3,IT 104,bala,dept4,Marketing

## Note: Delete the last blank line in the above file

1. **Create folder and copy file into hadoop**

hadoop fs -mkdir -p /user/sqoop/spexport

hadoop fs -copyFromLocal -f empinfo /user/sqoop/spexport

## In MySQL create tables and stored procedure

mysql -u root -p password: root

use custdb;

create table empinfo(empid int, empname varchar(20));

create table deptinfo(deptid varchar(10),deptname varchar(20)); delimiter //

create procedure sp\_insert\_empdeptinfo (IN pid int, IN pname varchar(20), IN pdeptid varchar(10), IN pdeptname varchar(20))

BEGIN

INSERT INTO empinfo(empid, empname) VALUES(pid, pname);

INSERT INTO deptinfo(deptid,deptname) values(pdeptid, pdeptname);

END // delimiter ;

## Export the data from the file to the mysql stored procedure to load data into two tables using SQOOP

sqoop export --connect jdbc:mysql://localhost/custdb --username root --password root --call sp\_insert\_empdeptinfo --export-dir /user/sqoop/spexport -m 1

## In mysql,select both the table empinfo and deptinfo to see the records

select \* from empinfo; select \* from deptinfo;

Importing data into Hive

sqoop import --connect jdbc:mysql://localhost/fleetdb \

--username root --password root \

--table driver \

-m 3 --split-by driverId \

--target-dir '/user/fleetdb/' --delete-target-dir \

--hive-import \

--create-hive-table \

--hive-table fleetdb.driver;

[hduser@Inceptez ~]$ hadoop fs -ls /user/hive/warehouse

Found 10 items

drwxr-xr-x - hduser supergroup 0 2019-06-20 07:55 /user/hive/warehouse/\_tmp.txnrecord\_es

drwxr-xr-x - hduser supergroup 0 2019-06-24 10:17 /user/hive/warehouse/\_tmp.txnrecord\_es\_id

drwxr-xr-x - hduser supergroup 0 2019-03-30 04:05 /user/hive/warehouse/custdb.db

drwxr-xr-x - hduser supergroup 0 2019-03-26 23:16 /user/hive/warehouse/customerall

drwxr-xr-x - hduser supergroup 0 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db

drwxr-xr-x - hduser supergroup 0 2019-03-29 20:49 /user/hive/warehouse/retail.db

drwxrwxr-x - hduser supergroup 0 2019-03-24 21:13 /user/hive/warehouse/retail1.db

drwxr-xr-x - hduser supergroup 0 2019-06-20 07:47 /user/hive/warehouse/txnrecord

drwxr-xr-x - hduser supergroup 0 2019-06-20 07:50 /user/hive/warehouse/txnrecord\_es

drwxr-xr-x - hduser supergroup 0 2019-06-24 10:17 /user/hive/warehouse/txnrecord\_es\_id

Even though we give target-dir, the data is imported into the database folder created in the warehouse folder.

Lets see inside the table folder.

[hduser@Inceptez ~]$ hadoop fs -ls /user/hive/warehouse/fleetdb.db/driver

Found 3 items

-rw-r--r-- 1 hduser supergroup 636 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00000

-rw-r--r-- 1 hduser supergroup 652 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00001

-rw-r--r-- 1 hduser supergroup 617 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00002

When we again run the same command, the data is appended to the same folder “driver” but as new files.

[hduser@Inceptez ~]$ hadoop fs -ls /user/hive/warehouse/fleetdb.db/driver

Found 6 items

-rw-r--r-- 1 hduser supergroup 636 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00000

-rw-r--r-- 1 hduser hadoop 636 2019-06-25 23:52 /user/hive/warehouse/fleetdb.db/driver/part-m-00000\_copy\_1

-rw-r--r-- 1 hduser supergroup 652 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00001

-rw-r--r-- 1 hduser hadoop 652 2019-06-25 23:52 /user/hive/warehouse/fleetdb.db/driver/part-m-00001\_copy\_1

-rw-r--r-- 1 hduser supergroup 617 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00002

-rw-r--r-- 1 hduser hadoop 617 2019-06-25 23:52 /user/hive/warehouse/fleetdb.db/driver/part-m-00002\_copy\_1

**Below are the major commands used in the –hive-import..**

| **Sqoop Command Option** | **Description** |
| --- | --- |
| --hive-home <directory> | Overrides $HIVE\_HOME. |
| --hive-import | Imports tables into Hive using Hive's default delimiters if none are explicitly set. |
| --hive-overwrite | Overwrites existing data in the Hive table. |
| --create-hive-table | Creates a hive table during the operation. If this option is set and the Hive table already exists, the job will fail. Set to false by default. |
| --hive-table <table\_name> | Specifies the table name to use when importing data into Hive. |
| --hive-drop-import-delims | Drops the delimiters \n, \r, and \01 from string fields when importing data into Hive. |
| --hive-delims-replacement | Replaces the delimiters \n, \r, and \01 from strings fields with a user-defined string when importing data into Hive. |
| --hive-partition-key | Specifies the name of the Hive field on which a sharded database is partitioned. |
| --hive-partition-value <value> | A string value that specifies the partition key for data imported into Hive. |
| --map-column-hive <map> | Overrides the default mapping from SQL type to Hive type for configured columns. |

Importing data into HBase

sqoop import --connect jdbc:mysql://localhost/fleetdb --username root --password root \

--table driver -m 3 --split-by driverId \

--hbase-table fleetdb \

--column-family driver \

--hbase-row-key driverId ;

**How can you import large objects like BLOB and CLOB in Sqoop?**  
The direct import function is not supported by Sqoop in case of CLOB and BLOB objects. Hence, if you have to import large purposes, you can use JDBC based imports. This can be done without introducing the direct argument of the import utility.

**How can we import data from RDBMS to Mainframes?**

Using Sqoop import-mainframe command

**Scoop supports data transfer to:**

Sqoop supports data imported into following services:

* HDFS
* Hive
* Hbase
* Hcatalog
* Accumulo
* Mainframes

**Role of JDBC driver in sqoop setup? Is the JDBC driver enough to connect the sqoop to the database?**

Sqoop needs a connector to connect the different relational databases. Almost all Database vendors make a JDBC connector available specific to that Database, Sqoop needs a JDBC driver of the database for interaction.  
*No, Sqoop needs JDBC and a connector to connect a database.*

**How do we ignore the header when export to RDBMS from HDFS**

By using “ ignore n lines”

Sqoop merge tool:

Used to merge two datasets.

[ambari-qa@mauve1 ~]$ hadoop fs -ls /apps/hive/warehouse/student

Found 2 items

-rwxrwx--- 1 ambari-qa hadoop 26 2017-02-08 23:13 /apps/hive/warehouse/student/part-m-00000

-rwxrwx--- 1 ambari-qa hadoop 26 2017-02-09 21:51 /apps/hive/warehouse/student/part-m-00000\_copy\_1

sqoop merge --new-data /apps/hive/warehouse/student/part-m-00000

--onto /apps/hive/warehouse/student/part-m-00000\_copy\_1

--target-dir /tmp/sqoop\_merge

--jar-file /tmp/sqoop-ambari-qa/compile/9062c87c959e4090dcec5995a439b514/TIME.jar

--class-name TIME

--merge-key TIME