SQOOP COMMANDS

Anytime during this exercise, if you need help on sqoop queries, use sqoop help option
$sqoop --help
$sqoop import --help

Import into HDFS – Database level operations

— list databases

```
$ sqoop list-databases
--connect "jdbc:mysql://quickstart.cloudera:3306"
--username retail_dba
--password cloudera
```

— import all tables from db to HDFS

```
sqoop import-all-tables -m 12
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--as-textfile
--warehouse-dir=/user/cloudera/sqoop_import/
```

Formats: supported are **avro**, **text** and **binary**
–as-textfile, –as-avrodatafile, –as-sequencefile

-m or –num-mappers: Used to define number of threads per table. **By default, there are 4 mappers**
BoundingValsQuery: Used to figure out number of buckets based on number of mappers.

— Import all tables from RDBMS with compression and hive table creation

```
$sqoop import-all-tables \
 --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
 --username retail_dba \
 --password cloudera \
 --hive-import \
 --hive-overwrite \
 --create-hive-table \
 --hive-database sqoop_import \
 --compress \
 --compression-codec org.apache.hadoop.io.compress.SnappyCodec \
 --outdir java_files
```

compress and compression-codec: is used to compress ingested files
out-dir: is used to store some sqoop internal java files
–hive-import and create-hive-table: used to import into hive warehouse and create hive tables on ingested tables
–hive-overwrite – overwrites the data in existing table, if not mentioned then it will append to the existing data in the table

Import into HDFS – Table level operations

— Import a single table from sqoop

```
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--target-dir /user/cloudera/departments
```

–table: mention table name
–target-dir: location where table data is copied
Note: If '-m' option is not given then default number of mappers=4
Note: For every table import sqoop will use min and max of primary key (in boundingvalquery) and divide the records into number of buckets as specified

* Disadvantage: With above query is that if there are some outliers in the data then data will be unevenly spread across mappers with some mappers taking heavy load and some less load

— overwrite boundary query to redefine the distribution

```
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--target-dir /user/cloudera/departments
--boundary-query "select min(department_id), max(department_id) from departments where department_id <> 8000"
```

NOTE : In some cases this query is not the most optimal(in case of outliers), so you can specify any arbitrary query returning two numeric columns using --boundary-query argument.

— import specific columns from a table

```
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--target-dir /user/cloudera/departments --boundary-query "select min(department_id), max(department_id) from departments where department_id <> 8000"
--columns department_id,department_name
```

— import a table using specific query

```
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
```

--target-dir /user/cloudera/departments
--table departments
--boundary-query "select min(department_id), max(department_id) from departments where department_id <> 8000"
--columns department_id,department_name
--query "select * from departments"

* –query and –table are mutually exclusive

Remember, the table must have a primary key when you import it to Hadoop. If it does not have a primary key, it will fail.

— import a table without primary key
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments_nopk
--target-dir /user/cloudera/departments

* This will error out as sqoop cannot split the records if there is no primary key.

Solution - In this case we should give either '-m 1' or '–split-by '

$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments_nopk
--target-dir /user/cloudera/departments
--m 1

OR

$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments_nopk
--target-dir /user/cloudera/departments
--split-by department_id

— import data by joining the source table
$ sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera

--query "select * from orders join order_items on orders.order_id=order_items.order_item_order_id where \$CONDITIONS"
--split-by order_id
--target-dir /user/cloudera/order_join
--where "orders.order_id <> 0"

*** –table-name cannot be given with –query**

* $CONDITIONS is required because sqoop will append conditions from –where otherwise 'true' (if no condition given)

* –split-by is given because there is no primary_key on the joined dataset

— import into HIVE Tables

$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--hive-home /user/hive/warehouse
--hive-import
--hive-overwrite
--hive-table sqoop_import.departments

* –hive-home is optional as it is the default value

* –hive-table should include db name followed by table name OR include –hive-database to have dbname separate

* There are two ways to import data into hive tables:
    1. create the table and then import into the existing table via –hive-table(above query)
    2. create table while importing itself via –create-hive-table

* Hive import will first download data into the temp dir (i.e, home dir of user /user/cloudera/) and then loads into the hive table, hence make sure the dir with the table name is deleted in your home directory

---

Incremental Load

---

* In Incremental Loads –
Before importing we connect to log table or log file to check for the delta condition (using sqoop eval or IO API) and then do import and update the log table/file after import is successful so that next incremental/delta load can look at it

* Incremental Load can be done in two ways –
    1. using where argument
    2. use out of the box incremental options –incremental, –check-column and –last-value

#Option-1

```
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--append
--target-dir /user/cloudera/sqoop_import/departments/
--where "department_id > 7"
```

* –append and –where works together in incremental loads. If –append not given then it will error out

#Option-2

```
$ sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--append
--target-dir /user/cloudera/sqoop_import/departments/
--check-column department_id
--incremental append
--last-value 7
```

–append is required in this case as well

–check-column : columns against which delta is evaluated
–last-value: last values from where data has to be imported
–incremental: append/lastmodified

* –incremental: append – Used when there are only inserts into the sql table (NO UPDATES)
* –incremental: lastmodified – Used when there are inserts and updates to the SQL table. For this to use we should have date column in the table and –last-value should be the timestamp

---

Export data to a MySQL database from HDFS using Sqoop

— Export HDFS data into new SQL table

```
sqoop export
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table order_items_export
--export-dir /user/cloudera/sqoop_import/order_items
```

* –export-dir is an option to specify external directory to load the data from hdfs into mysql table

* How number of threads/mappers work in export?
In import based on number of mappers('-m 12') sqoop will issue that many queries and imports data from mysql table into the cluster as RDBMS has that capability.
But in export, it uses HDFS distributed data blocks to divide the blocks among the threads ('–num-mappers 12') and starts uploading the data

— Update/Merge HDFS data into existing SQL table

```
$ sqoop export
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--export-dir /user/cloudera/sqoop_import/departments_export/
--batch
--update-key department_id
--update-mode allowinsert
```

* –update-key is the primary_key/unique_key against which the update will happen. There has to be a primary key on the table for the above query to work otherwise all records will be inserted (duplicate records). If there is composite key then give comma separated columns

* –update-mode : updateonly/allowinsert
updateonly – It updates the existing record/s and DOES NOT insert new record (DEFAULT MODE), all new records will be ignored. So without passing –update-mode argument, records can only be updated but new records cannot be inserted.

allowinsert – It can updates existing records and also inserts new records

* Without –update-key and –update-mode, it works only as insert mode.

Change the delimiter and file format of data during import using Sqoop

---

— Change import delimiters on plain HDFS dir

```
$ sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--target-dir /user/cloudera/departments_enclosed
--enclosed-by \" --fields-terminated-by \| --lines-terminated-by \\n --escaped-by \, --null-string \\N --null-non-string -1
```

* –enclosed-by: It encloses every field in the data with this character
* –escaped-by: Used to escape any special characters in the data (like , in csv can cause issue with total number of cols in a record)
* –fields-terminated-by: field separator
* –lines-terminated-by: line separator

* –null-string: Replace null in string columns
* –null-non-string: Replace null in non-string(int, double etc) columns
* Default values are Uses MySQL's default delimiter set: fields: , lines: \n escaped-by: \ optionally-enclosed-by: ' [These can be used with explicit arg –mysql-delimiters or don't give any args with respect to delimiters and formats]

— Change import delimiters on hive tables

Sqoop import using –hive-import options will import the data using default hive delimiters as fields: CTRL+A and lines: \n

```
$sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
 --password cloudera
--table departments
--hive-home /user/hive/warehouse
--hive-import --hive-overwrite
--hive-table sqoop_import.departments_test
--create-hive-table
```

— Change export delimiters

All the delimiters in HDFS input in export are appended with –input

* –input-enclosed-by: It encloses every field in the data with this character
* –input-escaped-by: Used to escape any special characters in the data (like , in csv can cause issue with total number of cols in a record)
* –input-fields-terminated-by: field separator
* –input-lines-terminated-by: line separator
* –input-null-string: Replace null in string columns
* –input-null-non-string: Replace null in non-string(int, double etc) columns

But if we are used non-default SQL delimiters when we imported the data and wanted to use same imported directory in export then we have to use above-to-above arguments as well as those delimiters will be stored in the out-dir (java-files) in the imported dir

```
$ sqoop export
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
 --username retail_dba
--password cloudera
--table departments_test
--export-dir /user/hive/warehouse/sqoop_import.db/departments_test/
--input-fields-terminated-by \\001
--input-lines-terminated-by '\n'
--input-null-string NULL
 --input-null-non-string -1
```

— file format of data during import

```
$ sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
```

--table departments
--target-dir /user/cloudera/departments
--as-sequencefile
–as-sequencefile: will store data in binary format

$ sqoop import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--target-dir /user/cloudera/departments
--as-avrodatafile

–as-avrodatafile will import schema into the user home dir along with the data into the target dir.

- Schema represents the table structure, columns and datatypes. It is generated with convention sqoop_import_.avsc
  - hive> Create external table departments_avro ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' stored as inputformat 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat' outputformat 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat' location '/user/cloudera/departments/' tblproperties('avro.schema.url'='/user/cloudera/departments.avsc');
- Export have nothing to do with file formats

Sqoop Jobs and Sqoop Merge

This is used to define pre-defined job with all the required parameters for the purpose of reuse
$ sqoop job
--create import_job
-- import
--connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username retail_dba
--password cloudera
--table departments
--target-dir /user/cloudera/departments

* — import \ [there should be space between — and import]
$sqoop job --list -> will list all the existing sqoop jobs
$sqoop job --show –> will show the job details and definition
$sqoop job --exec –> To run the job

— Merge
sqoop merge
--merge-key department_id
--new-data
--new-data /user/cloudera/sqoop_merge/departments_delta
--onto /user/cloudera/sqoop_merge/departments
--target-dir /user/cloudera/sqoop_merge/staging

--class-name departments.java
--jar-file /tmp/sqoop-cloudera/compile/e11d28e872acd71c103d33fbf81ec5c7/departments.jar

* now remove the old dir '/user/cloudera/sqoop_merge/departments'
hdfs dfs -rm -R /user/cloudera/sqoop_merge/departments
* rename dir '/user/cloudera/sqoop_merge/staging' to '/user/cloudera/sqoop_merge/departments'
hdfs dfs -mv /user/cloudera/sqoop_merge/staging /user/cloudera/sqoop_merge/departments

References:

https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html
https://www.youtube.com/channel/UCakdSIPsJqiOLqylgoYmwQg

Reference: https://tekmarathon.com/2016/12/21/sqoop-cheat-sheet/