# **SQOOP**

## **Introduction:**

* Started by Cloudera
* **SQ**L + HAD**OOP** 🡪 **SQOOP**
* Sqoop is mainly used to export and import data between RDBMS and Hadoop

HADOOP

RDBMS

HBASE

HIVE

## **Connecting to Database:**

* Connection URL
* Username and password of Database
* Driver Class Name
* Client Jar

## **SQOOP IMPORT:**

### **Verifying MySQL Data with SQOOP**

#### List Databases Command

* This Command will give the details about the databases available.
* Hadoop is not required to start to verify this command.
* This command is usually used to test database connection
* Similar to list-databases we have other commands like
* list-databases
* list-tables

sqoop list-databases \

--connect "jdbc:mysql://localhost:3306" \

--username root \

--password hadoop

#### Eval Command:

* Sqoop-eval command is used to execute user defined queries using query argument

sqoop eval \

--connect "jdbc:mysql://localhost:3306" \

--username root \

--password hadoop \

--query "select count(1) from order\_items"

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

sqoop eval \

--connect "jdbc:mysql://localhost:3306" \

--username root \

--password hadoop \

-e “INSERT INTO employee VALUES(1207,‘Raju’,‘UI dev’,15000,‘TP’)”

#### Options File:

* The Command line arguments that do not change can be put in an Options file.
* This is a text file and usually user specifies Connection Url, User name and password in this file.
* Options file has to be stored in a local directory.

# Options file for Sqoop import

# specifies the tool being invoked

import

# Connect parameter and value

--connect

jdbc:mysql://localhost/db

# Username parameter and value

--username

foo

#

# Remaining options should be specified in the command line.

#

sqoop eval \

--options-file

/home /work/sqoop\_inputs/connection\_details.txt \

--query "show tables"

### **SQOOP Import**

#### Sqoop import-all-tables

* For import-all-tables we use warehouse-dir argument
* Import-all is used to import all the tables that are available in the specified **warehouse-dir**
* **exclude-tables** argument can be used to exclude few tables available in the database

sqoop import-all-tables \

--connect "jdbc:mysql://localhost:3306/sandeep" \

--username=root \

--password=hadoop \

--warehouse-dir=/sqoop/import-all-data/sandeep.db \

--exclude-tables customers,order\_items,orders,products

#### Sqoop import

* For importing a single table we will go for sqoop import command instead of sqoop import-all-tables
* Difference between sqoop import and sqoop import-all-tales is that with import-all-tables, all the tables available in the database can be imported. With sqoop-import, we can import only a specific table into HDFS
* We use **target-dir** argument in sqoop import instead of warehouse-dir (target-dir is used to specify the location of HDFS where data will be imported)

Sqoop import \

--connect "jdbc:mysql://localhost:3306/sqoop" \

--username=root \

--password=hadoop \

--table pet \

--as-textfile \

--target-dir=/sqoop/import-data/sqoop.db/pet\_m \

-m 1

#### Sqoop storage options

* We can specify the type of extension in ‘- -as-’ argument in sqoop to store the data in HDFS

Eg: **--as-textfile**

* Sqoop supports the following storage options

1. Text File (**--as-textfile**)
2. Sequence File (**--as-sequencefile**)
3. Avro Data File (**--as-avrodatafile**)
4. Parquet File (**--as-parquetfile**)

#### Sqoop: Controlling Parallelism

* Sqoop internally converts every sqoop import or sqoop import-all-tables command into a java program and runs it as a Map Reduce program
* By Default, the number of mappers in a sqoop job in 4
* Mappers always run in parallel. So, all the imports always work in parallel.
* When performing parallel imports, sqoop tries to identify a criterion by which it can split the workload.
* By Default sqoop identifies a primary-key column (if present) in a table and uses it as a splitting column.
* The low and high values for the splitting column (primary key) are retrieved from the database, and the map tasks operate on evenly-sized components of the total range
* If actual values in the split by field are not uniformly distributed across its range, then this can result in unbalanced tasks
* If primary key is not present in the table, then we should explicitly specify the field with which we want to split the data by using **--split-by** argument

sqoop import \

--connect "jdbc:mysql://localhost:3306/sqoop" \

--username=root \

--password=hadoop \

--table pet \

--as-sequencefile \

--target-dir=/sqoop/import-data/sqoop.db/pet\_split\_by \

--**split-by name**

* We can control the number of mappers by using any of the following arguments
* **-m 1**
* **--num-mappers-1**
* With –num-mappers-1 argument, we can specify the number of mappers with which the program can run.
* This value works a degree of parallelism in RDBMS. We can specify a value greater than 1 as well in –num-mappers argument.
* If we want to specify only single mapper, then we can directly go with **–m 1**

Sqoop import \

--connect "jdbc:mysql://localhost:3306/sqoop" \

--username=root \

--password=hadoop \

--table pet \

--as-textfile \

--target-dir=/sqoop/import-data/sqoop.db/pet\_m \

-m 1

#### Sqoop: Overriding Type Mapping

* We can override default mapping provided by sqoop using the parameter

**--map-column-java**

sqoop import \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--table cities \

--map-column-java id=Long

* If we have to specify multiple columns then we can use,

--map-column-java id=Long, course=string, subject=string …

#### Sqoop: Encoding NULL Values

* Sqoop encodes database null values with ‘NULL’ string constant
* If there is a field in database which does not have a value, then that is imported with the value ‘NULL’ by sqoop
* We can overcome this by using the parameter

**--null-string (for VARCHAR2, VARCHAR, CHAR, NCHAR datatypes)**

**--null-non-string (for all other data types)**

* Internally, the values specified in the **--null(-non)-string** parameters are encoded as a string constant in the generated Java code. You can take advantage of this by specifying any arbitrary string using octal representation without worrying about proper encoding.
* If you want to use \N to encode missing values, then you need to specify \\N on the command line; \ is a special escape string character in Java that will be interpreted by the compiler.
* Your shell will try to unescape the parameters for you, so you need to enclose those parameters in single quotes ('). Using double quotes (") will cause your shell to interpret the escape characters, changing the parameters before passing them to Sqoop.

sqoop import \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--null-string '\\N' \

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

### **SQOOP Incremental Import**

For Transferring Deltas of data, sqoop offers to do incremental imports. For example, we have scenario where there is a table which has about 10k rows of data till 25th Dec and using sqoop import we already have all that data in HDFS. Now, there are another 5k rows which got inserted into the same table by 31st Dec. In this scenario, we need not import only the additional 5k rows of data that is there in the table. Sqoop Incremental import allows users to import only the remaining 5k rows instead of all 15k rows

#### Sqoop: Incremental Append

* We can activate the Incremental import by using the parameter

**--incremental append**

* Along with incremental append we have to use another 2 parameters

**--check-column**

**--last-value**

* Check column indicates a column that should be checked for newly appended data
* Last Value indicates the last value of column specified in the check column parameter that is successfully imported into Hadoop
* The limitation here is that we should know the Last value that is successfully imported into HDFS

sqoop import \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table visits \

--incremental append \

--check-column id \

--last-value 1

* When running in incremental mode, sqoop always prints the value of the last imported row allowing users to easily make a note of the last value

#### Sqoop: Incremental import last modified

* The incremental append mode lets us import only those rows which are additionally inserted into the table. If there are any existing records which got updated in the database, then incremental append mode will not capture those records
* We can achieve this by using incremental lastModified parameter

**--incremental lastModified**

* The incremental mode lastModified requires a column holding a date value containing information as to when the last row was imported into HDFS
* So, in Check Column parameter we have specify a date field (usually last\_update\_date) and last value parameter should have date and time stamp.

**--check-column**

**--last-value**

* Internally, the lastmodified incremental job consists of 2 standalone Map Reduce jobs
* The 1st job will import only updated records and stores into a temp directory in HDFS
* The 2nd job will import the newly inserted records and it will also merge the data of the updated records to give the final output

sqoop import \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table visits \

--incremental lastmodified \

--check-column last\_update\_date \

--last-value "2013-05-22 01:01:01"

#### Sqoop: Sqoop Job Commands

* One of the drawbacks of incremental import feature is that it requires users to remember the last value mentioned in the check column parameter
* We can overcome this problem by through sqoop metastore that allows us to retain all job definitions and save all parameter values for later use.
* We can achieve this by using Sqoop create argument

**Sqoop job --create myjob**

* The above command will ensure that all the parameters mentioned in the import command will be saved. We can access these jobs either from command line or from job tracker url which will list out all the parameter values of the specified job

sqoop job \

--**create myjob1** \

-- import \

--connect "jdbc:mysql://localhost:3306/sqoop" \

--username=root \

--password=hadoop \

--query="select \* from student where id <= 30 and \$CONDITIONS" \

--target-dir /sqoop/data/sqoop.db/student \

--append \

--num-mappers 1 \

--fields-terminated-by '|' \

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

--check-column "id" \

--incremental append \

--last-value 20 \

--outdir java\_files

* Some of the Sqoop Job Commands are,

**sqoop job --list**

**sqoop job --show myjob1**

**sqoop job --exec myjob1**

**sqoop job --delete myjob1**

### **SQOOP Free Form Query Import**

#### Sqoop: Free Form Query Import

* Sqoop will allow to import any data using a query
* Instead of specifying **--table** argument, we can specify **--query** parameter to specify the data to be imported
* Free Form Query import cannot be used in conjunction with **--warehouse-dir** parameter
* Table import is always faster than the equivalent free form query import because sqoop can’t use database catalogue to fetch the metadata.
* In addition to the **--query** parameter we need to specify --**split-by** parameter with the column that should be useful to slice the data into multiple parallel tasks
* To help sqoop transfer multiple chunks of data in parallel, we need to specify **$CONDITIONS** in the where clause of the query. Sqoop will automatically substitute this placeholder with required conditions based on the parameters specified in the query.
* While **$CONDITIONS** can be skipped by forcing sqoop to import with only 1 mapper using --num-mappers 1 parameter, this will have a severe performance impact
* If your query needs more than a few seconds in order to start sending data, it might not be suitable for the free-form query import. If this is the case, you can always run the expensive query once prior to Sqoop import and save its output in a temporary table. Then you can use table import to transfer the data into Hadoop.

sqoop import \

--connect "jdbc:mysql://localhost:3306/kalyan" \

--username=root \

--password=hadoop \

--query="select \* from orders join order\_items on orders.order\_id

= order\_items.order\_item\_order\_id where \$CONDITIONS" \

--target-dir /sqoop/join-data/kalyan.db/order\_join \

--split-by order\_id \

--num-mappers 1

#### Sqoop: Boundary Query

* Sqoop takes a very long time to fetch the min and max values of the column specified in the split-by parameter
* We can specify any valid query to fetch the min and max values for the column mentioned in split-by parameter using boundary-query parameter

**--boundary-query**

* In order to partition data into multiple slices and import them in parallel, sqoop needs to identify the min and max values of the column mentioned in split-by parameter
* In case of table import, sqoop will identify min and max based on the primary key value of the table to be imported. In most of the cases, primary key will have unique values. So, when it slices the data based on primary key, there will not be any issue.
* In case of Free Form Query import, we import data from multiple tables at once based on the query mentioned query argument. So, sqoop will convert the mentioned query in the query argument into a sub query and try to identify the min and max values of the column mentioned in split-by argument.
* Internally sqoop will treat it as ‘Select min(col), max(col) from $YOUR\_QUERY’ where $YOUR\_QUERY means the query mentioned in query argument. This will put significant overload on the database and the performance of the command will be very slow.
* With boundary-query argument, we can limit the import to specific set of records so that import will be much faster.

sqoop import \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--query 'SELECT normcities.id, \

countries.country, \

normcities.city \

FROM normcities \

JOIN countries USING(country\_id) \

WHERE $CONDITIONS' \

--split-by id \

--target-dir cities \

--boundary-query "select min(id), max(id) from normcities"

#### Sqoop: Importing Duplicate Columns

* During normal table import, all the columns of a single table will have unique names. So, there will not be any duplicate column names in a single query import.
* But while doing a free form query import, since we will be joining multiple tables, there is a possibility of having multiple columns with the same column name.
* When we encounter this issue, sqoop will give an error message,

‘Imported Failed: Duplicate Column identifier specified: 'id'’

* To overcome this issue we have use aliases for each column that we are selecting in the query of query argument. We have to ensure that each column has unique value in the select query the respective query argument.

## **SQOOP EXPORT**

### **SQOOP Export**

#### Sqoop: Data Transfer from Hadoop to RDBMS

* Sqoop Export command enables user to send data that is piled up in HDFS from Hive and Map Reduce Jobs into RDBMS
* Sqoop will transfer data into RDBMS in the form of INSERT statements
* Sqoop Fetches table’s metadata from RDBMS in the export command. So, the table mentioned in the table argument must be present in the database prior to executing sqoop export command
* The only mandatory condition is that there should not be any constraint violations like unique constraint violation

sqoop export \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--export-dir cities

* **--table** argument signifies the table to which the data should be exported
* **--export-dir** argument specifies the directory in HDFS that contains source data

#### Sqoop: Export Batch

* Sqoop Export command works well, but it creates separate INSERT statements for each row. So, it is very slow
* Instead we can specify an additional parameter with which data is exported faster. The additional parameter being,

**--batch**

* Batch parameter creates insert statements for multiple rows at a time instead of having separate insert statements for each row something like,

**Insert into table\_name values (…), (…),(…),(…),(…),……..**

* Not all databases support multiple row inserts in a single insert statement. Only Oracle, MYSQL and PostgreSQL support this feature.
* So, this parameter should be used based on the database that we are trying to export

sqoop export \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--export-dir cities \

--batch

#### Sqoop: Export Statements per statement

* Another option to insert the data from Hadoop to database is to specify number of records that will be used for each insert statement using the property

**-** **Dsqoop.export.records.per.statement**

sqoop export \

-Dsqoop.export.statements.per.statement=10 \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--export-dir cities

#### Sqoop: Export records per transaction

* We can also specify number of records that can be inserted in a single insert statement by using the property,

**-** **Dsqoop.export.records.per.transaction**

* The Default values can vary from connector to connector

sqoop export \

-Dsqoop.export.records.per.transaction=10 \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--export-dir cities

#### Sqoop: Export All or nothing

* We can use staging table to first export data into a temporary table before inserting the records into the real table. The stating table can specified by the parameter,

**--staging-table**

* When using a staging table, sqoop will first export all the data into staging table first rather than the table mentioned in table parameter.
* Sqoop will then start a new transaction to move data from staging table to the main table if and only if all the parallel tasks are successfully completed

sqoop export \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--staging-table staging\_cities

#### Sqoop: Export Update-key

* If we have a scenario where we have already exported data into HDFS and after export is completed, there is a change in the data, then we can update that data using update-key parameter.

**--update-key col\_name1, col\_name2, …..**

* All the columns that are specified in update-key parameter will be used in where clause of the update query generated.
* If we want to do update only then we can specify as updateonly in update-mode parameter

**--update-mode updateonly**

sqoop export \

--connect "jdbc:mysql://localhost:3306/kalyan1" \

--username root \

--password hadoop \

--table departments \

--export-dir /sqoop/kalyan.db/departments\_stage \

--batch \

--outdir java\_files \

-m 1 \

--update-key department\_id \

--update-mode updateonly

#### Sqoop: Export Update and Insert at same time

* We can activate upsert mode (allowing insert and update in a single command) by using allowinsert in the parameter update-mode

**--update-mode allowinsert**

* This mode will ensure that the existing rows which needs to updated get updated and new rows get inserted

sqoop export \

--connect "jdbc:mysql://localhost:3306/kalyan1" \

--username root \

--password hadoop \

--table departments \

--export-dir /sqoop/kalyan.db/departments\_stage \

--batch \

--outdir java\_files \

-m 1 \

--update-key department\_id \

--update-mode allowinsert

#### Sqoop: Export using Stored Procedures

* If the database that we are exporting our data into, already has an inbuilt solution to ingest the data, then we can directly the respective stored procedure

**--call populate\_payments**

sqoop export \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--call populate\_payments

#### Sqoop: Export columns

* By Default, sqoop assumes that the table that we are exporting data into has the same number and ordering of columns
* The parameter **columns** is used to specify either re-ordering of columns or only when a subset of columns are available in the input files

**--columns c1, c2, c3**

* We must ensure that the columns that are not being exported either support inclusion of NULL values or has a default value attached to it in the database.

sqoop export \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--columns country, city

## **SQOOP HIVE INTEGRATION**

### **SQOOP Export**

#### Sqoop: Hive Import

* We can directly import RDBMS data into hive using sqoop by hive-import parameter

**--hive-import**

* Biggest advantage of using sqoop for populating values into tables of hive is that it can automatically populate the metadata.
* If the table does not exist in Hive, then sqoop will automatically create the table using metadata fetched from the table or query. For this we have to use create-hive-table argument

**--create-hive-table**

* If the table already exists, then sqoop will import the data into the existing table.
* If we are creating a new table, then sqoop will convert the data types of each columns of the source table into a type compatible with hive.
* During hive import, sqoop will first do a normal hdfs import and store it at a temporary location. After this, sqoop generates 2 queries, one for table creation and another for loading data into the table from temporary location
* We can specify the temporary location in either –warehouse-dir or –target-dir parameter.

sqoop import-all-tables \

--connect "jdbc:mysql://localhost:3306/sqoop" \

--username=root \

--password=hadoop \

--hive-import \

--create-hive-table \

--warehouse-dir=/sqoop/hive\_data/sqoop.db \

--num-mappers 1 \

--outdir java\_files

#### Sqoop: Hive Overwrite

* If the Hive table already exists and has data, then sqoop will append newly imported data.
* We can change this behaviour by using hive-overwrite parameter which will truncate the data in the existing table and overwrite with the new data.

**--hive-overwrite**

sqoop import \

--connect "jdbc:mysql://localhost:3306/kalyan" \

--username=root \

--password=hadoop \

--table orders \

--fields-terminated-by '|' \

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

--hive-import \

**--hive-overwrite** \

**--hive-table myorders1** \

--outdir java\_files

#### Sqoop: Hive Partition

* Sqoop supports hive partitioning. We have to specify 2 parameters, hive-partition-key and hive-partition-value.

**--hive-partition-key**

**--hive-partition-value**

* Hive-partition-key contains the column name and hive-partition-value contains the desired value.
* Sqoop mandates that the partition column should only be of string data type.
* Sqoop does not accept a column name as a value for the parameter hive-partition-value.

sqoop import \

--connect jdbc:mysql://mysql.example.com/sqoop \

--username sqoop \

--password sqoop \

--table cities \

--hive-import \

--hive-partition-key day \

--hive-partition-value "2013-05-22"