Sqoop Additional Reading

IMPORTANT

Copyright Infringement and Illegal Content Sharing Notice

Concept-1

default import

By default, Sqoop will import a table named orders to a directory named orders inside your home directory in HDFS. For example, if your username is someuser, then the import tool will write to /user/someuser/orders/(files)

```
sqoop-import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table orders
```

Note: we are not specifying the target-dir or warehouse-dir
Also just like we import a table we can import a view also.

Concept-2

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.

If you want to import the results of a query in parallel, then each map task will need to execute a copy of the query, with results partitioned by bounding conditions inferred by Sqoop. Your query must include the token $CONDITIONS which each Sqoop process will replace with a unique condition expression. You must also select a splitting column with --split-by.

Note: If you are issuing the query wrapped with double quotes ("), you will have to use \$CONDITIONS instead of just $CONDITIONS to disallow your shell from treating it as a shell variable.

Example 1:

```
sqoop-import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--query 'select * from orders where $CONDITIONS AND order_id > 50000' \
--target-dir /data/orders3 \
--split-by order_id
```

Example 2:

```
sqoop-import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--query "select * from orders where \$CONDITIONS AND order_id >
50000" \
--target-dir /data/orders2 \
--split-by order_id
```

Note: The facility of using free-form query in the current version of Sqoop is limited to simple queries where there are no ambiguous projections and no OR conditions in the WHERE clause. Use of complex queries such as queries that have sub-queries or joins leading to ambiguous projections can lead to unexpected results.

```
[cloudera@quickstart ~]$ sqoop-import \
> --connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
> --username root \
> --password cloudera \
> --query "select * from orders where \$CONDITIONS AND order_id > 50000" \
> --target-dir /data/orders2 \
> --split-by order_id
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
20/04/14 18:05:24 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
20/04/14 18:05:24 WARN tool.BaseSqoopTool: Setting your password on the command-li
e is insecure. Consider using -P instead.
20/04/14 18:05:24 INFO manager.MySQLManager: Preparing to use a MySQL streaming re
ultset.
20/04/14 18:05:24 INFO tool.CodeGenTool: Beginning code generation
20/04/14 18:05:24 INFO manager.SqlManager: Executing SQL statement: select * from
rders where  (1 = 0)  AND order_id > 50000
20/04/14 18:05:24 INFO manager.SqlManager: Executing SQL statement: select * from
rders where  (1 = 0)  AND order_id > 50000
```

concept-3

Direct import

Controlling the Import Process
By default, the import process will use JDBC which provides a reasonable cross-vendor import channel. Some databases can perform imports in a more high-performance fashion by using database-specific data movement tools. For example, MySQL provides the mysqldump tool which can export data from MySQL to other systems very quickly. 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. But can be used for very basic things only.

example:

```
sqoop-import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table orders \
--target-dir /data/orders \
--direct
```

Concept 4:

Importing Data Into Hive

Sqoop's import tool's main function is to upload your data into files in HDFS. If you have a Hive metastore associated with your HDFS cluster, Sqoop can also import the data into Hive by generating and executing a CREATE TABLE statement to define the data's layout in Hive. Importing data into Hive is as simple as adding the --hive-import option to your Sqoop command line.

If the Hive table already exists, you can specify the --hive-overwrite option to indicate that existing table in hive must be replaced. After your data is imported into HDFS or this step is omitted, Sqoop will generate a Hive script containing a CREATE TABLE operation defining your columns using Hive's types, and a LOAD DATA INPATH statement to move the data files into Hive's warehouse directory.

Sqoop will by default import NULL values as string null. Hive is however using string \N to denote NULL values and therefore predicates dealing with NULL (like IS NULL) will not work correctly. You should append parameters --null-string and --null-non-string in case of import job or --input-null-string and --input-null-non-string in case of an export job if you wish to properly preserve NULL values. Because sqoop is using those parameters in generated code, you need to properly escape value \N to \\N:

sqoop import ... --null-string '\\N' --null-non-string '\\N'

The table name used in Hive is, by default, the same as that of the source table. You can control the output table name with the --hive-table option.

example:

```
sqoop-import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table orders \
--hive-import \
--hive-table orders_new \
--verbose
```

```
[cloudera@quickstart ~]$ sqoop-import \
> --connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
> --username root \
> --password cloudera \
> --table orders \
> --hive-import
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
20/04/14 18:23:19 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
20/04/14 18:23:19 WARN tool.BaseSqoopTool: Setting your password on the command-lin
e is insecure. Consider using -P instead.
20/04/14 18:23:19 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for outpu
t. You can override
20/04/14 18:23:19 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by,
etc.
20/04/14 18:23:19 INFO manager.MySQLManager: Preparing to use a MySQL streaming res
ultset.
20/04/14 18:23:19 INFO tool.CodeGenTool: Beginning code generation
20/04/14 18:23:19 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM
 `orders` AS t LIMIT 1
```

```
hive> show tables;
OK
emps
orders
Time taken: 0.017 seconds, Fetched: 2 row(s)
hive>
```

```
hive> show tables;
OK
emps
orders
Time taken: 0.017 seconds, Fetched: 2 row(s)
hive> select * from orders limit 10;
OK
1       2013-07-25 00:00:00.0   11599   CLOSED
2       2013-07-25 00:00:00.0   256     PENDING_PAYMENT
3       2013-07-25 00:00:00.0   12111   COMPLETE
4       2013-07-25 00:00:00.0   8827    CLOSED
5       2013-07-25 00:00:00.0   11318   COMPLETE
6       2013-07-25 00:00:00.0   7130    COMPLETE
7       2013-07-25 00:00:00.0   4530    COMPLETE
8       2013-07-25 00:00:00.0   2911    PROCESSING
9       2013-07-25 00:00:00.0   5657    PENDING_PAYMENT
10      2013-07-25 00:00:00.0   5648    PENDING_PAYMENT
Time taken: 0.417 seconds, Fetched: 10 row(s)
hive> \
```

```
20/04/14 18:28:55 WARN hive.TableDefWriter: Column order_date had to be cast to a l
ess precise type in Hive
20/04/14 18:28:55 DEBUG hive.TableDefWriter: Create_statement: CREATE TABLE IF NOT
EXISTS `orders_new` ( `order_id` INT, `order_date` STRING, `order_customer_id` INT,
 `order_status` STRING) COMMENT 'Imported by sqoop on 2020/04/14 18:28:55' ROW FORM
AT DELIMITED FIELDS TERMINATED BY '\001' LINES TERMINATED BY '\012' STORED AS TEXTF
ILE
20/04/14 18:28:55 DEBUG hive.TableDefWriter: Load statement: LOAD DATA INPATH 'hdfs
://quickstart.cloudera:8020/user/cloudera/orders' INTO TABLE `orders_new`
20/04/14 18:28:55 INFO hive.HiveImport: Loading uploaded data into Hive
20/04/14 18:28:55 DEBUG hive.HiveImport: Using in-process Hive instance.
20/04/14 18:28:55 DEBUG util.SubprocessSecurityManager: Installing subprocess secur
ity manager

Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-1
.1.0-cdh5.13.0.jar!/hive-log4j.properties
OK
Time taken: 1.885 seconds
Loading data to table default.orders_new
```

concept-5

Importing Data Into HBase

Sqoop supports additional import targets beyond HDFS and Hive. Sqoop can also import records into a table in HBase.

By specifying --hbase-table, you instruct Sqoop to import to a table in HBase rather than a directory in HDFS. Sqoop will import data to the table specified as the argument to --hbase-table. Each row of the input table will be transformed into an HBase Put operation to a row of the output table. The key for each row is taken from a column of the input. By default Sqoop will use the split-by column as the row key column. If that is not specified, it will try to identify the primary key column, if any, of the source table. You can manually specify the row key column with --hbase-row-key. Each output column will be placed in the same column family, which must be specified with --column-family.

If the target table and column family do not exist, the Sqoop job will exit with an error. You should create the target table and column family before running an import. If you specify --hbase-create-table, Sqoop will create the target table and column family if they do not exist, using the default parameters from your HBase configuration.

concept-6

Validate

Validate the data copied, either import or export by comparing the row counts from the source and the target post copy. Validation currently only validates data copied from a single table into HDFS and there are a lot of limitations.

example:

```
sqoop-import \
--connect jdbc:mysql://quickstart.cloudera:3306/retail_db \
--username root \
--password cloudera \
--table orders \
--target-dir /data/orders \
--validate
```

concept-7

Exports may fail for a number of reasons:

1. Loss of connectivity from the Hadoop cluster to the database (either due to hardware fault, or server software crashes)

2. Attempting to INSERT a row which violates a consistency constraint (for example, inserting a duplicate primary key value)

3. Attempting to parse an incomplete or malformed record from the HDFS source data

4. Attempting to parse records using incorrect delimiters

5. Capacity issues (such as insufficient RAM or disk space)

Note: If an export map task fails due to these or other reasons, it will cause the export job to fail. The results of a failed export are undefined. Each export map task operates in a separate transaction. Furthermore, individual map tasks commit their current transaction periodically. If a task fails, the current transaction will be rolled back. Any previously-committed transactions will remain durable in the database, leading to a partially-complete export.

Follow US

Trainer	Mr. Sumit Mittal
Phone	9108179578
Email	trendytech.sumit@gmail.com
Website	https://trendytech.in/courses/big-data-online-training/
LinkedIn	https://www.linkedin.com/in/bigdatabysumit/
Twitter	@BigdataBySumit
Instagram	bigdatabysumit
Facebook	https://www.facebook.com/trendytech.in/
Youtube	TrendyTech