Table of Contents [[hide](http://hadooptutorial.info/sqoop-interview-cheat-sheet/)]

* [Install sqoop](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Install_sqoop)
* [install jdbc drivers](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#install_jdbc_drivers)
  + [import](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#import)
* [import all](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#import_all)
* [mappers m1](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#mappers_m1)
* [Subset of Data where](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Subset_of_Data_where)
* [encoding null values](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#encoding_null_values)
* [sharing matastore](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#sharing_matastore)
* [install jdbc oozie](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#install_jdbc_oozie)
* [staging table](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#staging_table)
* [Problem](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Problem)
* [Imports into HBase take significantly more time than importing as text files in HDFS. Solution Create your HBase table prior to running Sqoop import, and instruct HBase to create more regions with the parameter NUMREGIONS. For example, you can create the HBase table cities with the column family world and 20 regions using the following command: hbase> create ‘cities’, ‘world’, {NUMREGIONS => 20, SPLITALGO => ‘HexString Split’} Sqoop](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Imports_into_HBase_take_significantly_more_time_than_importing_as_text_files_in_HDFS_Solution_Create_your_HBase_table_prior_to_running_Sqoop_import_and_instruct_HBase_to_create_more_regions_with_the_parameter_NUMREGIONS_For_example_you_can_create_the_HBase_table_cities_with_the_column_family_world_and_20_regions_using_the_following_command_hbasegt_create_8216cities8217_8216world8217_NUMREGIONS_gt_20_SPLITALGO_gt_8216HexString_Split8217_Sqoop)
* [Filters](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Filters)
* [Why Hbase?](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Why_Hbase)
* [Share this:](http://hadooptutorial.info/sqoop-interview-cheat-sheet/#Share_this)

**Install sqoop**

sudo yum install sqoop

1. sudo apt-get install sqoop
2. in sqoop-normal commnd prompt
3. sqoop config file—sqoop site.xml

**install jdbc drivers**

After you’ve obtained the driver, you need to copy the driver’s JAR file(s) into Sqoop’s lib/ directory. If you’re  
using the Sqoop tarball, copy the JAR files directly into the lib/ directory after unzipping the tarball. If you’re using packages, you will need to copy the driver files into the /usr/lib/sqoop/lib directory

**sqoop list-databases**



|  |  |
| --- | --- |
| 1  2 | connect jdbc:mysql://localhost/employees --username root |

**import**

sqoop import



|  |  |
| --- | --- |
| 1  2 | --connect jdbc:mysql://loacalhost/employees --username root table employees |

\*results are imported to hdfs under cloudera-employees

In employees directory 4 files are created

* part-m-00000
* part-m-00001
* part-m-00002
* part-m-00003

**import all**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | sqoop import-all-tables \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop  sqoop import-all-tables \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --exclude-tables cities,countries |

**mappers m1**

sqoop import



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | --connect jdbc:mysql://localhost/employees --username root --table employees -m1 --target-dir /user/cloudera/m1employees    part-m-00000-300024 recrds stored in one file(csv format comma seperated values)  import from 2  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 |

**Subset of Data  
where**

sqoop import



|  |  |
| --- | --- |
| 1  2 | --connect jdbc:mysql://localhost/employees --username root --table employees -m1 --where "gender='M'" --target-dir /user/cloudera/whereemployees |

part-m-00000-179973 records.  
When using the –where parameter, keep in mind the parallel nature of Sqoop transfers.  
Data will be transferred in several concurrent tasks. Any expensive function call will  put a significant performance burden on your database server. Advanced functionscould lock certain tables, preventing Sqoop from transferring data in parallel. This willadversely affect transfer performance. For efficient advanced filtering, run the filteringquery on your database prior to import, save its output to a temporary table and run  
Sqoop to import the temporary table into Hadoop without the –where parameter.

boundary queries



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | --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"  normcities.city \  text,avro,sequenceFile(binaryformat)    sqoop import --connect jdbc:mysql://localhost/employees --username root --table employees -m1 --where "emp\_no='10001'" --as-sequencefile --target-dir /user/cloudera/sequenceemployees |



|  |  |
| --- | --- |
| 1  2  3  4  5 | sqoop import --connect jdbc:mysql://localhost/employees --username root --table employees -m1 --where "emp\_no='10001'" --as-avrodatafile --target-dir /user/cloudera/avrodataemployees    part-m-00000  Obj avro.schema? {"type":"record","name":"employees","doc":"Sqoop import of employees","fields":[{"name":"emp\_no","type":["int","null"],"columnName":"emp\_no","sqlType":"4"},{" |



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | sqoop import --connect jdbc:mysql://localhost/employees --username root --table employees -m1 --where "emp\_no='10001'" --compress --target-dir /user/cloudera/gzipemployees    718b-normally  64b-after compression  part-m-00000.gz    Transferred 64 bytes in 24.2191 seconds |

By default, when using the –compress parameter, output fileswill be compressed using the GZip codec, and all files will end up with a .gz extension.  
You can choose any other codec using the –compression-codec parameter.

other compression codecs  
bzip2-.bz2



|  |  |
| --- | --- |
| 1  2  3 | sqoop import --compress \  --compression-codec org.apache.hadoop.io.compress.BZip2Codec |

As Sqoop delegates compression to the MapReduce engine, you need to make sure the compressed map output is allowed in your Hadoop configuration. For example, if in the mapred-site.xml file, the property  
mapred.output.compress is set to false with the final flag, then Sqoop won’t be able to compress the output files even when you call it with the –compress parameter.  
The selected compression codec might have a significant impact on subsequent processing.  
Some codecs do not support seeking to the middle of the compressed file without reading all previous content, effectively preventing Hadoop from processing the input files in a parallel manner. You should use a splittable codec for data that you’re planning to use in subsequent processing.

* Compression codecs
* Splittable Not Splittable
* BZip2, LZO GZip, Snappy



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | overiding type mapping  --map-column-java id=Long    sqoop import --map-column-java c1=Float,c2=String,c3=String  controlling parallelism  Sqoop by default uses four concurrent map tasks to transfer data to Hadoop    --num-mappers 10  or --m10 |

**encoding null values**

Sqoop encodes database NULL values using the null string constant. Your downstream processing (Hive queries, custom MapReduce job, or Pig script) uses a different constant for encoding missing values



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | --null-string '\\N' \  --null-non-string '\\N'    Incremental imports  sqoop import \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --table visits \  --incremental append \  --check-column id \  --last-value 1  incremental lastmodify  --incremental lastmodified \  --check-column last\_update\_date \  --last-value "2013-05-22 01:01:01"    create job  sqoop job \  --create visits \  -- \  import \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --table visits \  --incremental append \  --check-column id \  --last-value 0  And start it with the --exec parameter:  sqoop job --exec visits    sqoop job --list  You can remove the old job definitions that are no longer needed with the --delete  parameter, for example:  sqoop job --delete visits    renaming job  --mapreduce-job-name normcities |

**sharing matastore**

Sqoop’s metastore can easily be started as a service with the following command:  
sqoop metastore  
Other clients can connect to this metastore by specifying the parameter –meta-connect in the command line with the URL of this machine. For example, to create a new saved job in the remote metastore running on the host



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | mestastore.example.com, you can execute the following command:  sqoop job  --create visits \  --meta-connect jdbc:hsqldb:hsql://metastore.example.com:16000/sqoop \  -- \  import \  --table visits  ...  30 |  oozie  <workflow-app name="sqoop-workflow" xmlns="uri:oozie:workflow:0.1">  ...  <action name="sqoop-action">  <sqoop xmlns="uri:oozie:sqoop-action:0.2">  <job-tracker>foo:8021</job-tracker>  <name-node>bar:8020</name-node>  <command>import --table cities --connect ...</command>  </sqoop>  <ok to="next"/>  <error to="error"/>  </action>  <arg>import</arg>  <arg>--table</arg>  <arg>cities</arg>  <arg>--username</arg>  <arg>sqoop</arg>  <arg>--password</arg>  <arg>sqoop</arg> |

The content of each <arg> tag is considered to be one parameter regardless of how many spaces it contains; this is especially useful for entering queries as <arg>SELECT \* FROMcities</arg>, which is considered to be one single parameter. Having spaces inside oa <command> tag might not be obvious, especially when you’re using variables to parametrizeyour workflow. The preferred way to use Sqoop in Oozie is with <arg> tags.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | using property parameter in oozie---  <property>  <name>sqoop.export.statements.per.transaction</name>  <value>1</value>  </property> |

**install jdbc oozie**

Sqoop works correctly when executed from the command line, but in Oozie it cannot  
find the JDBC drivers.  
Solution  
You need to install the JDBC drivers into Oozie separately. You have two options: install  
the driver either into your workflow’s lib/ directory or into the shared action library  
location usually found at /user/oozie/share/lib/sqoop/.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Export  sqoop export \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --table cities \  --export-dir cities    export in batch  --table cities \  --export-dir cities \  --batch |

sqoop.export.records.per.statement tospecify the number of records that will be used in each insert statement:Finally, you can set how many rows will be inserted per transaction with the  
sqoop.export.statements.per.transaction property:



|  |  |
| --- | --- |
| 1  2  3  4 | sqoop export \  -Dsqoop.export.statements.per.transaction=10 \  --connect jdbc:mysql://mysql.example.com/sqoop \ |

**staging table**

kalyana13:You can use a staging table to first load data to a temporary table before making changes  
to the real table. The staging table name is specified via the –staging-table parameter.  
In the below example,

updating dataset



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | sqoop export \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --table cities \  --update-key id    UPDATE table SET c1 = ?, c3 = ? WHERE c2 = ? and c3 = ?  updating & inserting  Updating or Inserting at the Same Time-------------------  sqoop export \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --table cities \  --update-key id \  --update-mode allowinsert |

The ability to conditionally insert a new row or update an existing one is an advanced database feature known as upsert. This feature is not available on all database systems nor supported by all Sqoop connectors. Currently it’s available only for Oracle and nondirect MySQL exports.

stored procedure  
Using Stored Procedures—————-



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | sqoop export \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --call populate\_cities |

parameter. For example, when exporting into MySQL,  
Sqoop uses the following query:  
CALL populate\_cities(?, ?, ?)

import into hive



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | sqoop import \  --connect jdbc:mysql://mysql.example.com/sqoop \  --username sqoop \  --password sqoop \  --table cities \  --hive-import    --hive-import \  --map-column-hive id=STRING,price=DECIMAL  partitioned hive  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" |

Sqoop mandates that the partition column be of type STRING. The current implementation is limited to a single partition level. Unfortunately, you can’t use this feature if your table has more than one level of  partitioning (e.g., if you would like a partition by day followed by a partition by hour).  
delims You can instruct Sqoop to automatically clean your data using –hive-dropimport- delims, which will remove all \n, \t, and \01 characters from all string-based columns:  
If removing the special characters is not an option in your use case, you can take advantage of the parameter –hive-delims-replacement, which will accept a replacement string.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import to hbase  Importing Data into HBase----------------  --username sqoop \  --password sqoop \  --table cities \  --hbase-table cities \  --column-family world    Importing All Rows into HBase-------------  -Dsqoop.hbase.add.row.key=true \    Improving Performance When Importing into HBase--------------------- |

**Problem**

**Imports into HBase take significantly more time than importing as text files in HDFS.  
Solution  
Create your HBase table prior to running Sqoop import, and instruct HBase to create  
more regions with the parameter NUMREGIONS. For example, you can create the HBase  
table cities with the column family world and 20 regions using the following  
command:  
hbase> create ‘cities’, ‘world’, {NUMREGIONS => 20, SPLITALGO => ‘HexString  
Split’}  
Sqoop**

Sqoop will connect to your Hadoop cluster and submit a MapReduce job. Eachmapper of the job will then transfer a slice of the table’s data. As MapReduce executesmultiple mappers at the same time, Sqoop will be transferring data in parallel to achievethe best possible performance by utilizing the potential of your database server. Eachmapper transfers the table’s data directly between the database and the Hadoop cluster.  
To avoid becoming a transfer bottleneck, the Sqoop client acts as the overseer ratherthan as an active participant in transferring the data. This is a key tenet of Sqoop’s design.

**Filters**

HBase filters are a powerful feature that can greatly enhance your effectiveness when working with data stored in tables.

Most concrete filter classes are direct descendants of FilterBase, As CompareFilter-based filters add one more feature to the base FilterBase class,namely the compare() operation, it has to have a user-supplied operator type that defines how the result of the comparison is interpreted.

**Why Hbase?**

HBase is a distributed and scalable data store that supports row-level updates, rapid queries, and row-level transactions (but not multirow transactions).

hbase is good for oltp

HBase can be used like a key-value store, where a single key is used for each row to provide very fast reads and writes of the row’s columns or column families. HBase also keeps a configurable number of versions of each column’s values (marked by timestamps), so it’s possible to go “back in time” to previous values, when needed.