SQOOP Insight

Presented By
Siva Kumar Bhuchipalli

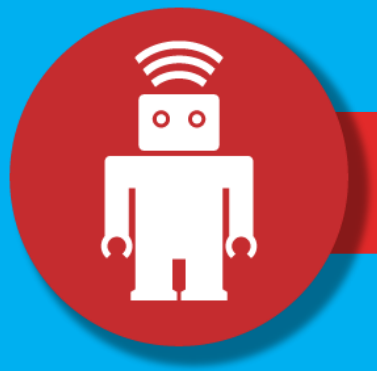
Contents

What is SQOOP?

- A command line interface
- For data import / export to Hadoop
- Uses Map jobs from Map Reduce
- Supports incremental loads
- Written in Java
- Licensed by Apache
- Uses plugins for new types of data source

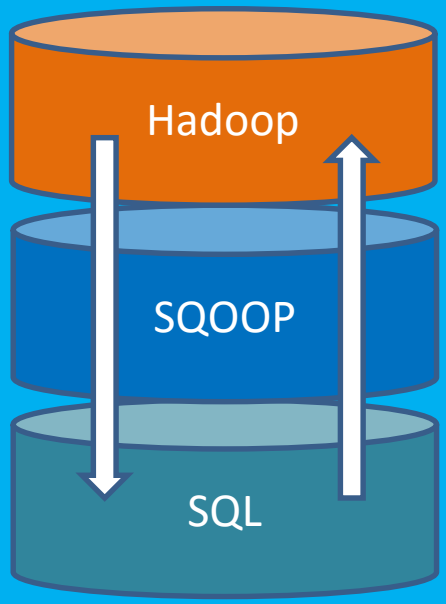
How SQOOP Works?

- Data sliced into partitions
- Mappers transfer data
- Data types determined via meta data
- Many data transfer formats supported
- i.e. CSV, SequenceFile, Parquet
- Can import into
  - Hive (use -- hive-import flag)
  - Hbase (use – hbase * flags)
  - Hcatalog

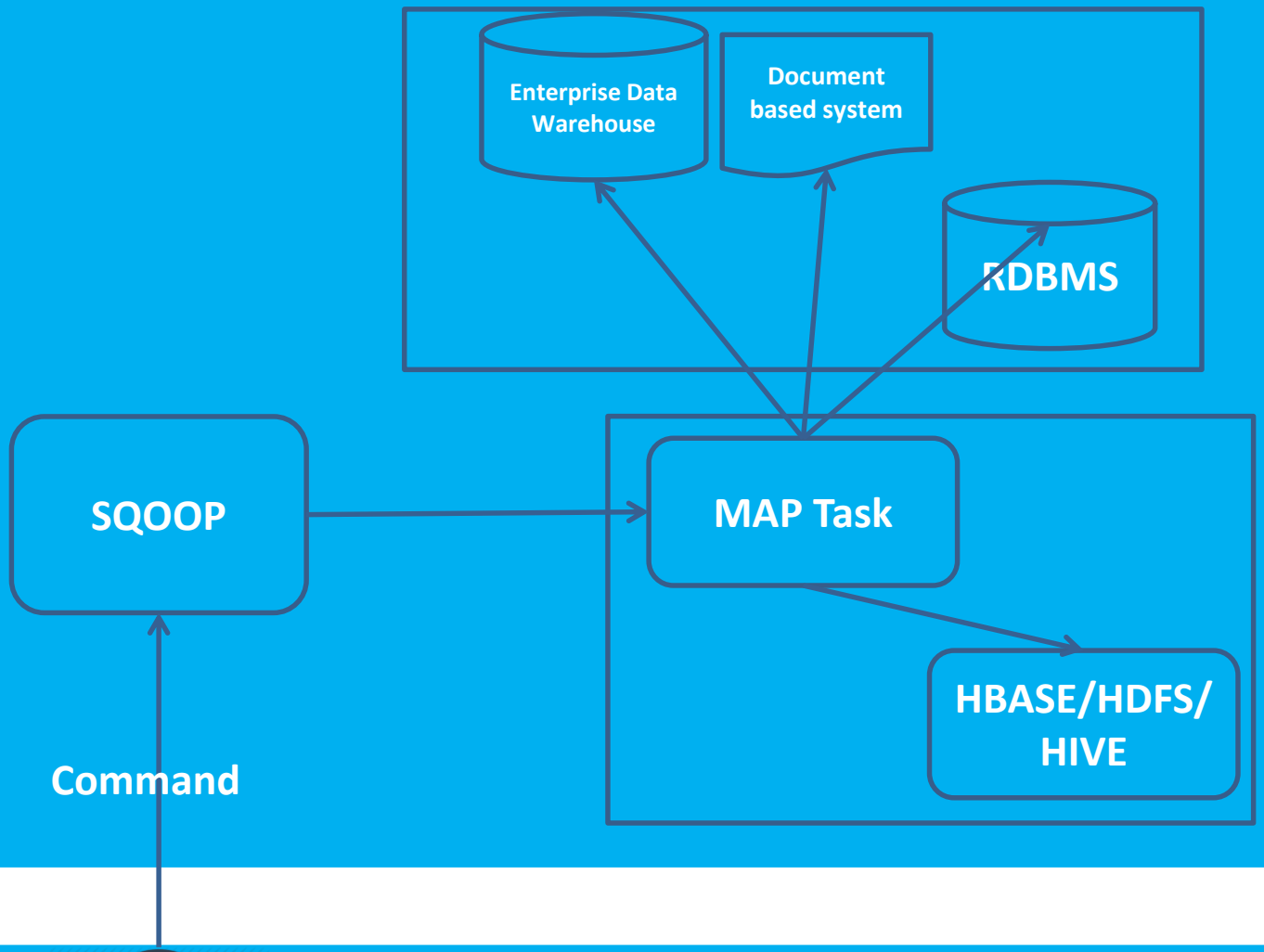


SQOOP Interfaces

- Get data from
  - -- Relational databases
  - -- data warehouses
  - -- NoSQL databases
- Load to HDFS and Hive
- Integrates with Oozie for scheduling



SQOOP Architecture



SQOOP Import

- Divide table into ranges using primary key max/min
- Create mappers for each range
- Mappers write to multiple HDFS nodes
- Creates text or sequence files
- Generates Java class for resulting HDFS file
- Generates Hive definition and auto-loads into HIVE

SQOOP Export

- Read files in HDFS directory via MapReduce
- Bulk parallel insert into database table

SQOOP features:

- Compatible with almost any JDBC enabled database
- Auto load into HIVE
- Hbase support
- Special handling for database Nulls
- Job management
- Cluster configuration (jar file distribution)
- WHERE clause support

SQOOP fast paths & plug ins

- Invoke MySQL dump, MySQL import for MySQL jobs
- Similar fast paths for PostgreSQL
- Extensibility architecture for 3rd parties (like Quest ☺)
- Teradata, Netezza, etc.

Importing Data - Lists Databases In Your MySQL Database.

[cloudera@quickstart ~]$ sqoop list-databases --connect jdbc:mysql://localhost/ --username root --password Cloudera

$ sqoop list-tables --connect jdbc:mysql://localhost/retail_db --username root --password cloudera

$ sqoop eval --connect jdbc:mysql://localhost/retail_db --username root --password cloudera --query 'select * from products limit 2‘

Importing Data In MySql Into HDFS

Replace "hadoop1-mySqlServer-node" with the host name of the node running MySQL server, replace login credentials and target directory.

[cloudera@quickstart ~]$ sqoop list-tables --connect jdbc:mysql://localhost/testdb1 --username root --password cloudera

SQOOP import --connect jdbc:mysql://<<mysql-server>>/employees --username myUID --password myPWD --table Employees -m 1 --target-dir /user/hadoop1/sqoop-mysql/employees

.

.

15/05/31 22:21:58 INFO mapreduce.ImportJobBase: Retrieved 300024 records

Import All Rows But Column Specific

```
$ Sqoop --table dept_emp \
--columns  "EMP_NO,DEPT_NO,FROM_DATE,TO_DATE" --as-textfile  -m 1 \
--target-dir /user/hadoop1/sqoop-mysql/DeptEmp
```

Import All Column, But Rows Specific Using Where Clause

```
$ Sqoop --table employees \
--where  "EMP_NO>499948"  --as-textfile  -m 1 \
--target-dir /user/hadoop1/sqoop-mysql/employeeGtTest
```

Import Free Form Query

```
$ sqoop import --connect jdbc:mysql://localhost/testdb1 --username root --password cloudera --query 'select e_id, e_name from emp where e_id > 10 and $CONDITIONS' -m 1 --target-dir /user/cloudera/sqoop/employee --delete-target-dir
```

Import Without Where Clause

```
$ sqoop \
--query ‘select EMP_NO, FIRST_NAME, LAST_NAME from employee
where $CONDITIONS’ –m 1 \
--target-dir /user/hadoop1/sqoop-mysql/employeeFrfrmQry2
```

Oracle Import Commands

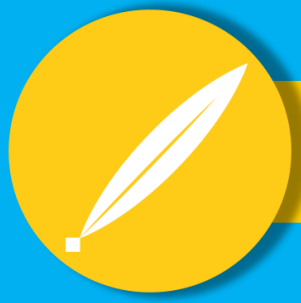
```
sqoop import \
--connect jdbc:oracle:thin:@trpherarm01:1535:VALDEVI1 \
--username NSP_GUEST \
--password NSP_GUEST \
--table NSP_V.TMPO_DDE_OUTPUT_FACTS \
--target-dir /user/jayara13/fact.txt

sqoop import \
--connect jdbc:oracle:thin:@trpherarm01:1535:VALDEVI1 \
--username NSP_GUEST \
--password NSP_GUEST \
--table NSP_V.TRDS_DATA_SCOPE_OUTPUT_FACT \
--target-dir /user/jayara13/TRDS_DATA_SCOPE_OUTPUT_FACT.txt

sqoop import --connect jdbc:oracle:thin:@trpherarm01:1535:VALDEVI1 \
--username NSP_GUEST --password NSP_GUEST \
--query 'select * from NSP_V.TRAG_OUTPUT_FACT WHERE $CONDITIONS'  \
--split-by OUF_ID --target-dir /user/jayara13/TRAG_OUTPUT_FACT.txt
```

EXPORT: Create Sample Table Employees

```
Mysql> CREATE TABLE Employees export (
Emp_no int(11) NOT NULL,
Birth_date date NOT NULL,
First_name varchar(14) NOT NULL,
Last_name varchar(16) NOT NULL,
Gender enum('M','F') NOT NULL,
Hire_date date NOT NULL,
PRIMARY KEY (emp_no)
);
```



Import Employees To HDFS To Demonstrate Export

```
$ sqoop  \
--query 'select EMP_NO, BIRTH_DATE,FIRST_NAME, LAST_NAME, GENDER, HIRE_DATE
from employee where  $CONDITIONS'  \
--split-by  EMP_NO \
--direct \
--target-dir /user/hadoop1/sqoop-mysql/Employees
```

EXPORT – Create A Stage Table

```
Mysql> CREATE TABLE Employees_exp_stg (
Emp_no int(11) NOT NULL, Birth_date date NOT NULL,
First_name varchar(14) NOT NULL,
Last_name varchar(16) NOT NULL,
Gender enum('M','F') NOT NULL,Hire_date date NOT NULL,
PRIMARY KEY (emp_no));
```

The Export Command

```
$ sqoop export \
--conect  jdbc:mysql://hadoop1-mysqlserver-node/employees  \
--username MyUID \
--password MyPWD \
--table employee_export \
--staging-table employees_exp_stg \
--clear-staging-table \
-m 4  \
--export-dir /user/hadoop1/sqoop-mysql/Employees
.
.
15/06/04 09:54:18 INFO manager.sqlManager: Migrated 300024 records from
'employees_exp_stg' to 'employees_export'
```

Results To Export

```
mysql> select * from employees_export limit 1;
+------------+----------------+----------------+---------------+-----------+------------------+
| emp_no  | birth_date   | first_name  | last_name  | gender  | hire_date       |
+------------+----------------+----------------+---------------+-----------+------------------+
| 200000    | 1960-01-11 |  Siva           | Bhuchipalli| M          | 1987-06-05    |
+------------+----------------+----------------+---------------+-----------+------------------+

mysql> select  count(*) from employees_export;
+------------+
| emp_no  |
+------------+
| 300024    |
+------------+

mysql> select  * from employees_exp_stg;
Empty set (0.00 sec)
```

Controlling the import parallelism (using 8 parallel tasks):

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
EMPLOYEES \ -m 8
```

Enabling the MySQL "direct mode" fast path:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
EMPLOYEES \ --direct
```

Storing data in SequenceFiles, and setting the generated class name to com.foocorp.Employee:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp -table
 EMPLOYEES \ --as-sequencefile
```

Specifying the delimiters to use in a text-mode import:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
 EMPLOYEES \
 --fields-terminated-by '\t' --lines-terminated-by '\n' \
 --optionally-enclosed-by '\"'
```

Importing the data to Hive:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
EMPLOYEES \
--hive-import
```

Importing only new employees:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
EMPLOYEES \
--where "start_date > '2010-01-01'"
```

Changing the splitting column from the default:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
 EMPLOYEES --split-by dept_id
```

Performing an incremental import of new data, after having already imported the first 100,000 rows of a table:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/somedb --table
sometable --where "id > 100000" --target-dir /incremental_dataset
 --append
```

Import all tables from the corp database:

```
$ sqoop import-all-tables --connect jdbc:mysql://db.foo.com/corp
```

A basic export to populate a table named bar:

```
$ sqoop export --connect jdbc:mysql://db.example.com/foo –table
 bar \ --export-dir /results/bar_data
```

Creating saved jobs is done with the --create action:

This operation requires a -- followed by a tool name and its arguments. The tool and its arguments will form the basis of the saved job. Consider:

```
$ sqoop job --create myjob -- import --connect jdbc:mysql://
example.com/db --table mytable
```

MERGE:

```
$ sqoop merge --new-data newer --onto older --target-dir merged --
jar-file datatypes.jar --class-name Foo --merge-key id
```

SQOOP EVAL

This allows users to preview their import queries to ensure they import the data they expect.

Select ten records from the employees table:

```
$ sqoop eval --connect jdbc:mysql://db.example.com/corp \ --query
 "SELECT * FROM employees LIMIT 10"
```

Insert a row into the foo table:

```
$ sqoop eval --connect jdbc:mysql://db.example.com/corp \ -e
"INSERT INTO foo VALUES(42, 'bar')"
```

SQOOP Job

Job is nothing but to save a sqoop command and execute n times when we require it. The job commands allows you to create and work with saved jobs . Saved jobs remembers the parameters used to specify a job, so they can be re-executed by invoking the job by its handle.

CREATE A JOB:

```
sqoop job --create myjob -- import --connect
jdbc:mysql://localhost/sqoop_export --username root -P --table
student_exported --target-dir /sqoop/test_job
```

List Job: It will show all the jobs.

```
Sqoop job --list
```

Inspect Job: It will show all the jobs.

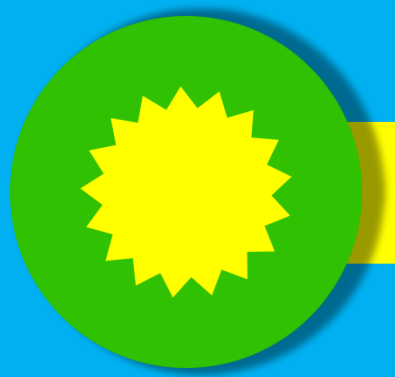
```
Sqoop job -show<jobname>
```

Delete Job: It will delete existing jobs.

```
Sqoop job -- delete <jobname>
```

Execute Job: it will execute the job.

```
Sqoop job –exec myjob
```



Command Arguments

As of Sqoop 1.4.5 version, Sqoop import command supports various number of arguments to import relational database tables into below tools or services.

- HDFS
- Hive
- HBase
- Hcatalog

There are specific arguments for these services, apart from

- Common arguments
- Control arguments
- Input parsing arguments
- Output line formatting arguments
- Incremental import arguments.

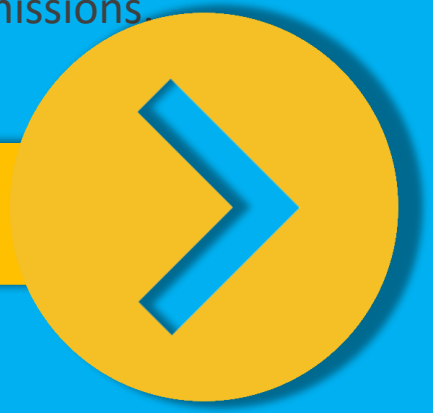
Lets discuss about each of these arguments with a brief explanation about each argument.

-connect <jdbc-uri>	To Specify JDBC connect string containing hostname or IP address (optionally port) followed by database name. It is mandatory argument. –connect jdbc:mysql:// localhost/ test_db In this example we are connecting to MySQL On localhost and database name is test_db
–connection-manager <class>	Specify connection manager class name It is optional. Ex: –connection-manager org.apache.sqoop.manager.GenericJdbcManager
–driver <class>	Manually specify JDBC driver class to use. Ex: –driver oracle.jdbc.OracleDriver
–username & (–password or -P)	We can specify the username and password of the RDBMS to retrieve the tables. Example: –username u1 –password 12345

Note

- We should not use the URL localhost in the connect string, if we intend to use Sqoop with a distributed Hadoop cluster. The connect string we supply will be used on data nodes throughout our cluster; if we specify the literal name localhost, each node will connect to a different database on their localhosts (or more likely, no database at all).
  Instead, we should use the full hostname or IP address of the database host that can be seen by all our remote nodes.

- The –password parameter is insecure, as other users may be able to read our password from the command-line arguments via the output of programs such as ps. The -P argument (prompts for user password) will read a password from a console prompt ,and is the preferred method of entering credentials. Credentials may still be transferred between nodes of the MapReduce cluster using insecure means. The most secure way is to use, –password-file <file containing the password> method. Set authentication password in this file on the users home directory with 400 permissions.



- The jar containing the driver class (ex: com.mysql.jdbc.Driver) should be copied into $SQOOP_HOME/lib directory otherwise exception will be thrown.

Below is the example sqoop import command covering all the arguments discussed above.

```
sqoop import
--connect jdbc:mysql://hostname_of_db/test_db \
--connection-manager org.apache.sqoop.manager.GenericJdbcManager  \
--driver com.mysql.jdbc.Driver \
--username u1 \
--password 12345 \
--table test_table
```

Control Argument

Argument	Description & Example
–as-textfile	Imports data as plain text file. It is the default option.
–as-sequencefile	Imports data to SequenceFiles
–columns <col,col,col…>	Specify Columns to import from RDBMS table
–table <table-name>	The table to import
-e, –query <statement>	It imports the results of query instead of a table.
–where <clause>	Import only the rows for which _clause_ is true. Example: –where “user_id > 400 AND hidden == 0”

–num-mappers <n> or -m <n>	Using this option we can speed the parallel copy process by setting the number of map tasks high. To set this, RDBMS table should have primary key.
–append	Append data to an existing HDFS dataset.
-z,–compress	Enable compression. By default uses GZip codec if not overridden below arg.
–target-dir <dir>	Explicit HDFS target directory for the import
–split-by (column-name)	Column of the table used to split the table for parallel import. By default sqoop will use query select min(<split-by>), max(<split-by>) from <table name> to find out boundaries for creating splits. By default, Sqoop will identify the primary key column (if present) in a table and use it as the splitting column.
–delete-target-dir	Delete the import target directory if it exists

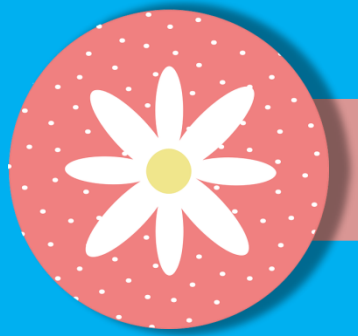
Note

- By default sqoop will run 4 map tasks.
- By default Sqoop uses JDBC channel to import, but with –direct option, sqoop uses high-performance direct channel but this is supported only in MySQL and PostgreSQL. We can split the output files based max size on HDFS, with –direct-split-sizeargument.
- When importing results of a query, we must specify destination directory with –target-dir.
- If we need import to be done in parallel we need to specify -m value greater than 1, so that multiple mappers can run in parallel but we must specify the split-by column or table should have primary key. Query must include the token $CONDITION which each Sqoop process will replace with a unique condition expression as shown in below example.

```
$ sqoop import \
  --query 'SELECT emp.*, usr.* FROM emp JOIN dep on (emp.id == usr.id) WHERE $CONDITIONS' \
  --split-by emp.id --target-dir /user/emp/join/
```

- -target-dir argument is mutually exclusive with –warehouse-dir. If the destination directory already exists in HDFS, Sqoop will refuse to import. If we use the –append argument, Sqoop will import data to a temporary directory and then rename the files into normal target directory in a manner that, it does not conflict with existing file names in that directory.

- Delimited text is the default import format but sqoop also supports sequence files and avro files. Text files cannot hold binary fields (such as database columns of type VARBINARY) and cannot distinguish between null values and String-based fields containing the value “null”. In this cases, we need to use sequence files or avro files.



- Sqoop supports large objects (BLOB and CLOB columns) also. By default, large objects less than 16 MB in size (this can be configured by –inline-lob-limit argument) are stored inline with the rest of the data. At a larger size, they are stored in files in the _lobs sub-directory of the import target directory.

```
sqoop import \
--connect jdbc:mysql://mysql.server.com/sqoop \
--table emp \
--null-string '\\N' \
--null-non-string '\\N'
```

Incremental Import Argument

Sqoop provides Incremental import which allows us to to retrieve only rows newer than some previously-imported set of rows by specifying the –incremental parameter. But it expects two more mandatory arguments, –check-column and –last-value.

Arguments	Description & Example
–incremental <type>	Valid import-types are 'append' or 'lastmodified'.
–check-column <col>	Source column to check for incremental change. (not be of type CHAR/ NCHAR/ VARCHAR/ VARNCHAR/ LONGVARCHAR/ LONGNVARCHAR) (suitable types are date , time , datetime , and timestamp)
–last-value (value)	Last imported maximum value in the incremental check column

Notes:

When RDBMS table is only getting new rows and the existing rows are not changed, then we need to use the append mode. When existing rows also being updated in addition to new rows then we need to use lastmodified incremental mode. Internally, the lastmodified incremental import fires two mapreduce jobs. The first job will import the delta of changed data similarly to normal import. This import job will save data in a temporary directory on HDFS. The second job will take both the old and new data and will merge them together into the final output, preserving only the last updated value for each row.

Example: **sqoop import **
**--connect jdbc:mysql://mysql.server.com/sqoop **
**--table employee **
**--incremental append **
--check-column id --last-value 100

HIVE Argument

Apart from importing RDBMS tables into HDFS files, Sqoop also support importing RDBMS tables directly into Hive tables and it is very simple that just using –hive-import argument to import command. Optionally,Sqoop will generate a Hive script containing a CREATE TABLE operation defining table columns using Hive's types, and a
LOAD DATA INPATH statement to move the data files into Hive's warehouse directory.

Arguments	Description
–hive-import	Import tables into Hive (Uses Hive's default delimiters if none are set.)
–map-column-hive <arg>	Override mapping for specific column to hive types.
–hive-table <table-name>	Sets the table name to use when importing to hive

–hive-partition-key <partition-key>	Sets the partition key to use when importing to hive
–hive-partition-value <partition-value>	Sets the partition value to use when importing to hive
–hive-database <database-name>	Sets the database name to use when importing to hive

Note:

- The table name used in Hive is, by default, the same as that of the source RDBMS table. We can change the output table name with the –hive-table option.
- If the Hive table already exists, we can override it with –hive-overwrite option.
- Hive Importing doesn't support –as-avrodatafile and –as-sequencefile clauses.

- By default, row delimiters in hive are \n and \r characters and column delimiters \001 characters, if source RDBMS tables contain these characters inside fields, then we need to use –hive-drop-import-delims option to drop those characters on import to give Hive-compatible text data, or –hive-delims-replacement option to replace those characters with a user-defined characters.
- Sqoop can import data for Hive into a particular partition by specifying the –hive-partition-key and –hive-partition-value arguments

Examples:

```
sqoop import --connect jdbc:mysql://localhost/testdb1 --username root --password cloudera --table emp -m 1 --target-dir /user/cloudera/sqoop/tmp_employee --delete-target-dir --hive-import --hive-table employee --map-column-hive 'e_id=BIGINT,e_name=STRING,salary=DOUBLE,designation=STRING,dept=STRING' --create-hive-table
```

HBase Arguments

Apart from HDFS and Hive, Sqoop also supports importing RDBMS tables into HBase tables. Each row of the input table will be transformed into an HBase Put operation to a row of the output table.

Arguments	Description
–column-family <family>	Sets the target column family for the import
–hbase-bulkload	Enables HBase bulk loading
–hbase-create-table	If specified, create missing HBase tables
–hbase-row-key <col>	Specifies which input column to use as the row key
–hbase-table <table>	Import to <table> in HBase

Note:

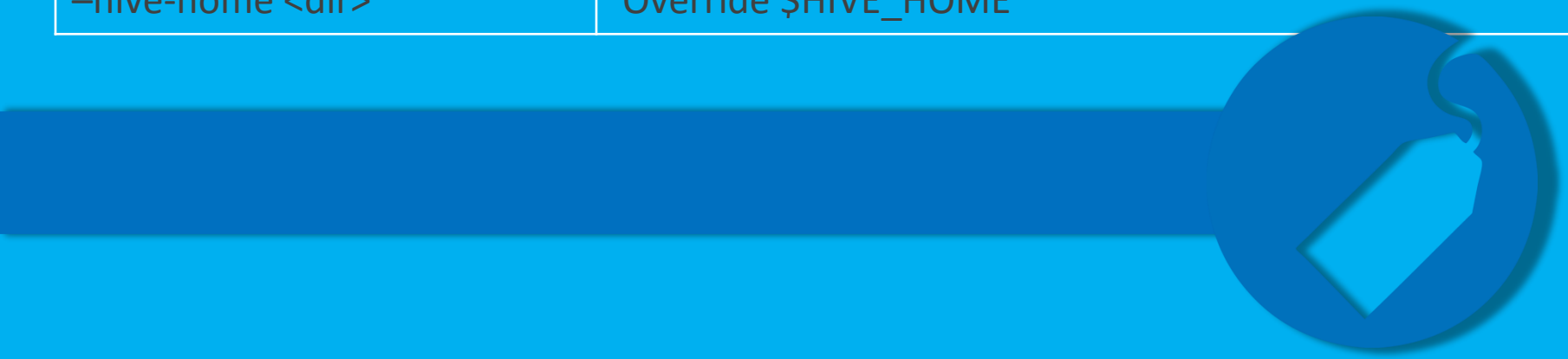
- Hbase row-key is decided by split-by column if it is mentioned, otherwise, sqoop will assign primary key of the source table. We can specify our own row-key with –hbase-row-key option.
- All output columns will be placed in the same column-family specified by –column-family.
- If the target table and column family do not exist, the Sqoop job will exit with an error, we can specify –hbase-create-table option to create the target table and column family if they do not exist.

- **sqoop import --connect jdbc:mysql://localhost/testdb1 --username root --password cloudera --table emp -m 1 --target-dir /user/cloudera/sqoop/tmp_employee --delete-target-dir --hbase-table emp1 --column-family cf1 --hbase-create-table --hbase-row-key e_id**

HCatalog Arguments

Recently HCatalog Importing is also supported by Sqoop.

Arguments	Description
–hcatalog-database <arg>	HCatalog database name, If not specified, the default database name default is used
–hcatalog-home <hdir>	Override $HCAT_HOME
–hcatalog-partition-keys <partition-key>	Sets the partition keys to use when importing to hive
–hcatalog-partition-values <partition-value>	Sets the partition values to use when importing to hive
–hcatalog-table <arg>	HCatalog table name
–hive-home <dir>	Override $HIVE_HOME



–hive-partition-key <partition-key>	Sets the partition key to use when importing to hive
–hive-partition-value <partition-value>	Sets the partition value to use when importing to hive
–map-column-hive <arg>	Override mapping for specific column to hive types.
–create-hcatalog-table	Create HCatalog tables before import, By default, HCatalog tables are assumed to exist.
–hcatalog-storage-stanza <arg>	HCatalog storage stanza for table creation

Notes:

- Using –hcatalog-database option without –hcatalog-table will result an error message.
- The presence of the –hcatalog-table option signifies that the import or export job is done using HCatalog tables, and it is a required option for HCatalog jobs
- The following Sqoop export and import options are not supported with HCatalog jobs.
  - ✓ –direct
  - ✓ –export-dir
  - ✓ –target-dir
  - ✓ –warehouse-dir
  - ✓ –append
  - ✓ –as-sequencefile
  - ✓ –as-avrofile

Thank You