# SQOOP

Sqoop is a tool designed to transfer data between Hadoop and relational databases or mainframes. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle or a mainframe into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS.

Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

### Why SQOOP?

#### Technical Challenges:

When we pull data from a table call transaction which might contain millions of data, using a SQLload,

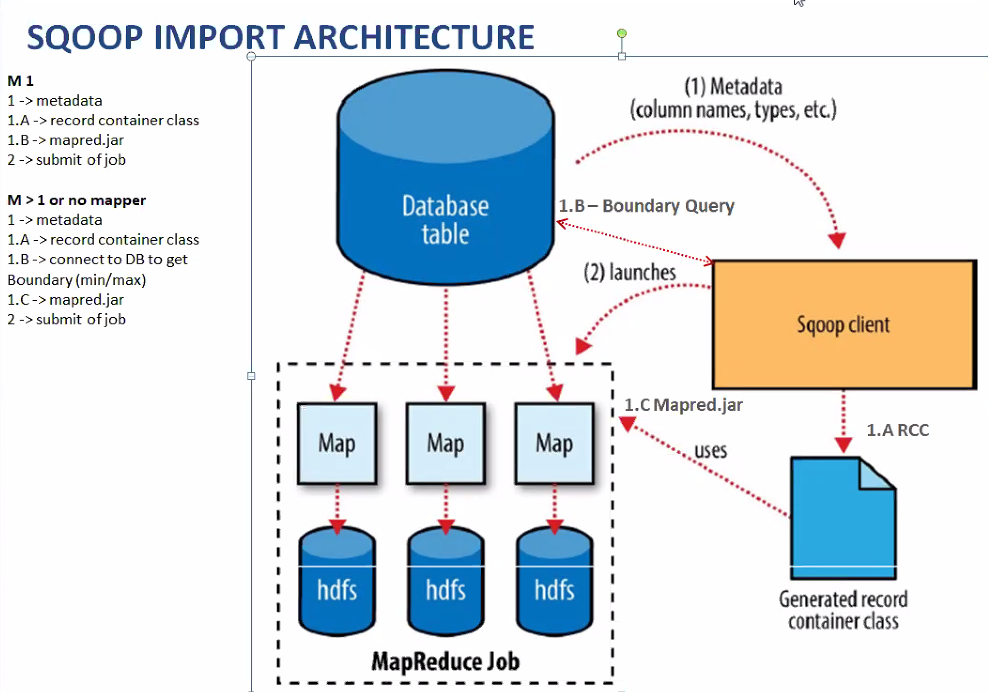
First Export 🡪 to file

Create a .sh file to load into HDFS into edge node using FTP/SFTP protocol

Ensure, transfer is complete or not

Data transfer happens using a single thread

# Sqoop import arch



* 1. Get meta data of database
  2. Record container class - create a java program having datatype convention data
  3. Mapred.jar – map reduce program – collection of data

1. Submitting of mapred.jar job to YARN
2. YARN architecture

## Sqoop Command Structure:

sqoop + toolname(import/export) + generic\_argument + tool\_specific\_args

sqoop list-tables --connect jdbc:mysql://localhost/custdb --username root --password root;

--direct 🡪 fetch in one shot using the source tool

--fetch size 🡪 how much data should be presented to Hadoop mappers at a time, default is 1000 rows at a time

When and Why to use single – and double -- ?

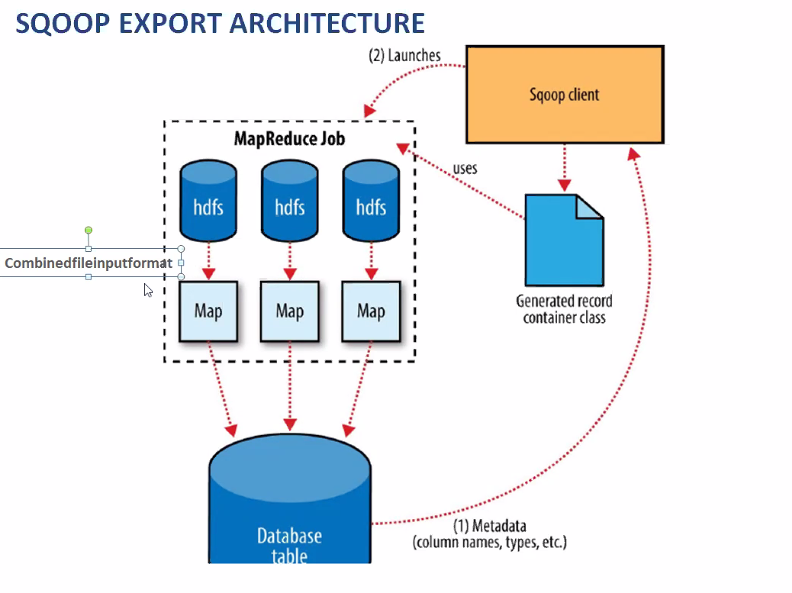
If we want to give more than one command then – is give

**Sqoop Import** – helps to import data from rdbms into Hadoop

First, Sqoop will connect to the database to fetch table metadata: the number of table columns, their names, and the associated data types. For example, for table cities, Sqoop will retrieve information about the three columns: id, country, and city, with int, VARCHAR, and VARCHAR as their respec‐ tive data types. Depending on the particular database system and the table itself, other useful metadata can be retrieved as well (for example, Sqoop can determine whether the table is partitioned or not). At this point, Sqoop is not transferring any data between the database and your machine; rather, it’s querying the catalog tables and views. Based on the retrieved metadata, Sqoop will generate a Java class and compile it using the JDK and Hadoop libraries available on your machine

Next, Sqoop will connect to your Hadoop cluster and submit a MapReduce job. Each mapper of the job will then transfer a slice of the table’s data. As MapReduce executes multiple mappers at the same time, Sqoop will be transferring data in parallel to achieve the best possible performance by utilizing the potential of your database server. Each mapper 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 rather than as an active participant in transferring the data. This is a key tenet of Sqoop’s design

### Export:

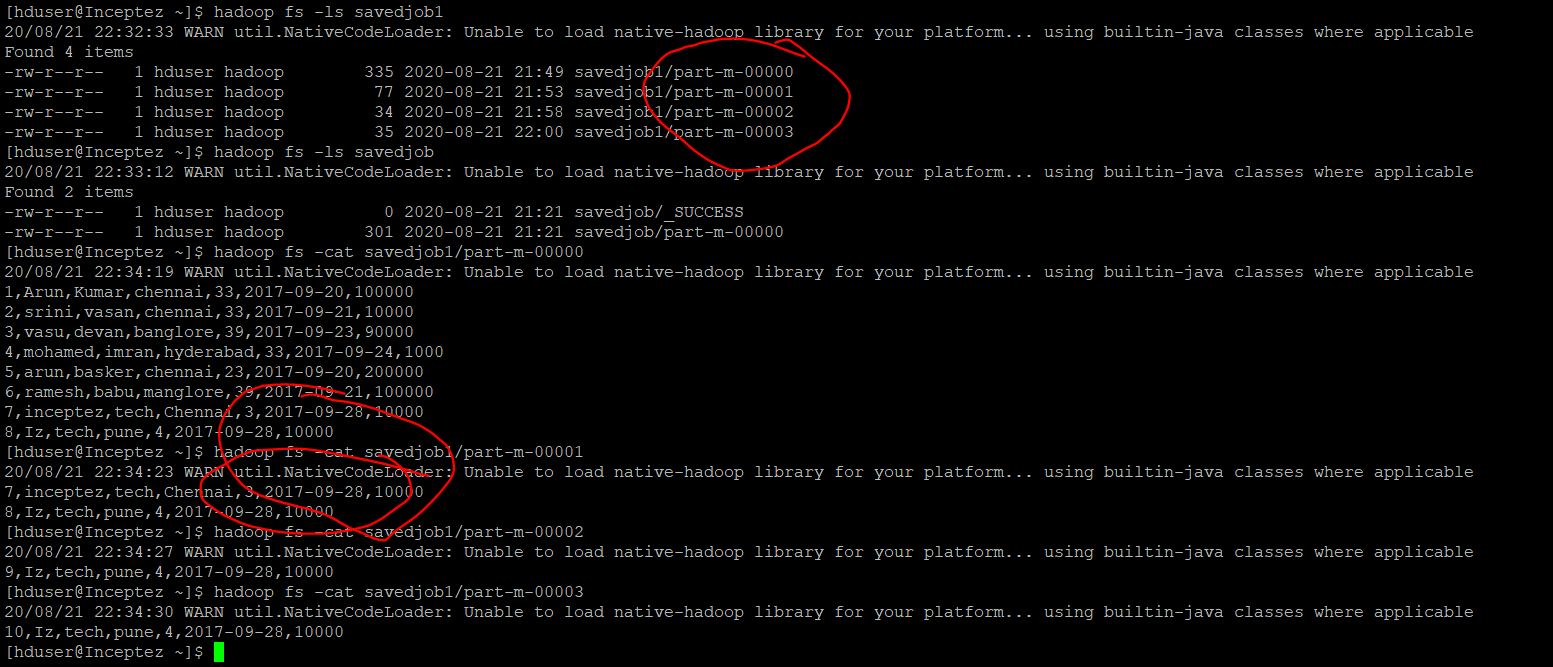


Cominedfileinputformat:

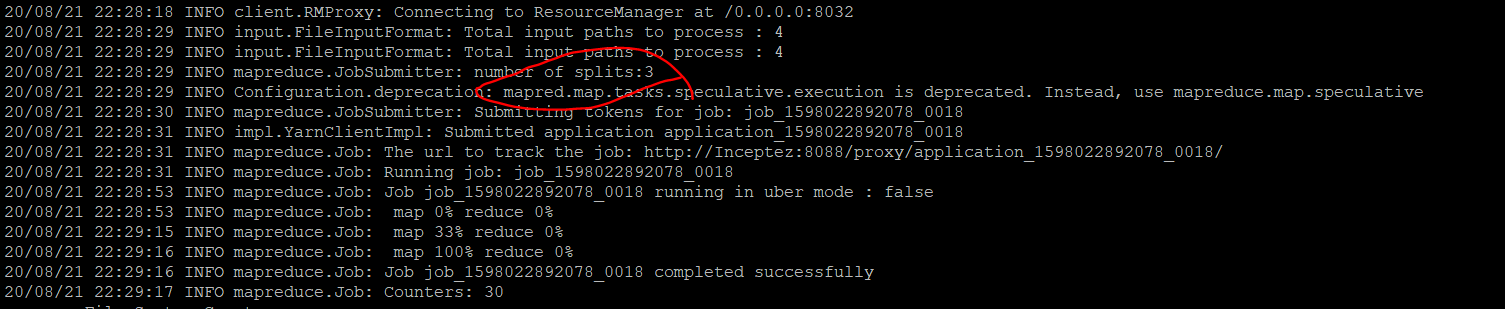
**sqoop export --connect jdbc:mysql://localhost/custdb --username root --password root --table customer\_hdfs --export-dir savedjob1**

When there are no input splits are mentioned, it calculates automatically; No of Input splits = no of blocks of data:

**HDFS:**

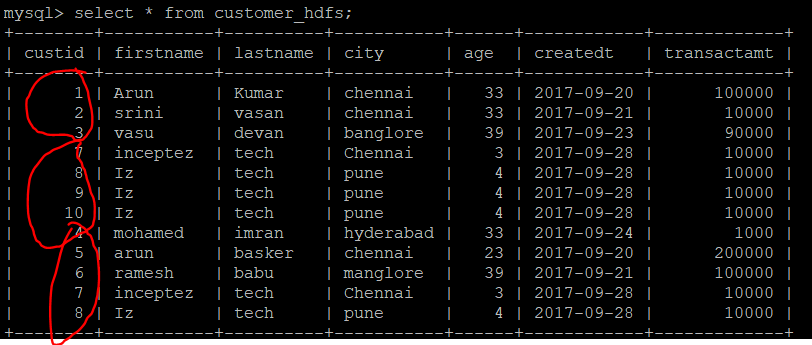


Sqoop:



We can also mention number of InputSplits in the command itself

sqoop export --connect jdbc:mysql://localhost/custdb --username root --password root –table customer\_hdfs -- export-dir savedjob1 -m 1

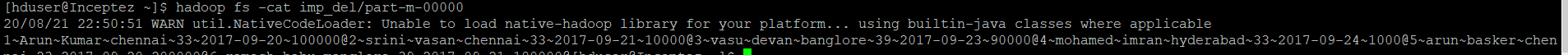


The data is not sorted 🡪 3 input splits worked separately to add data into database

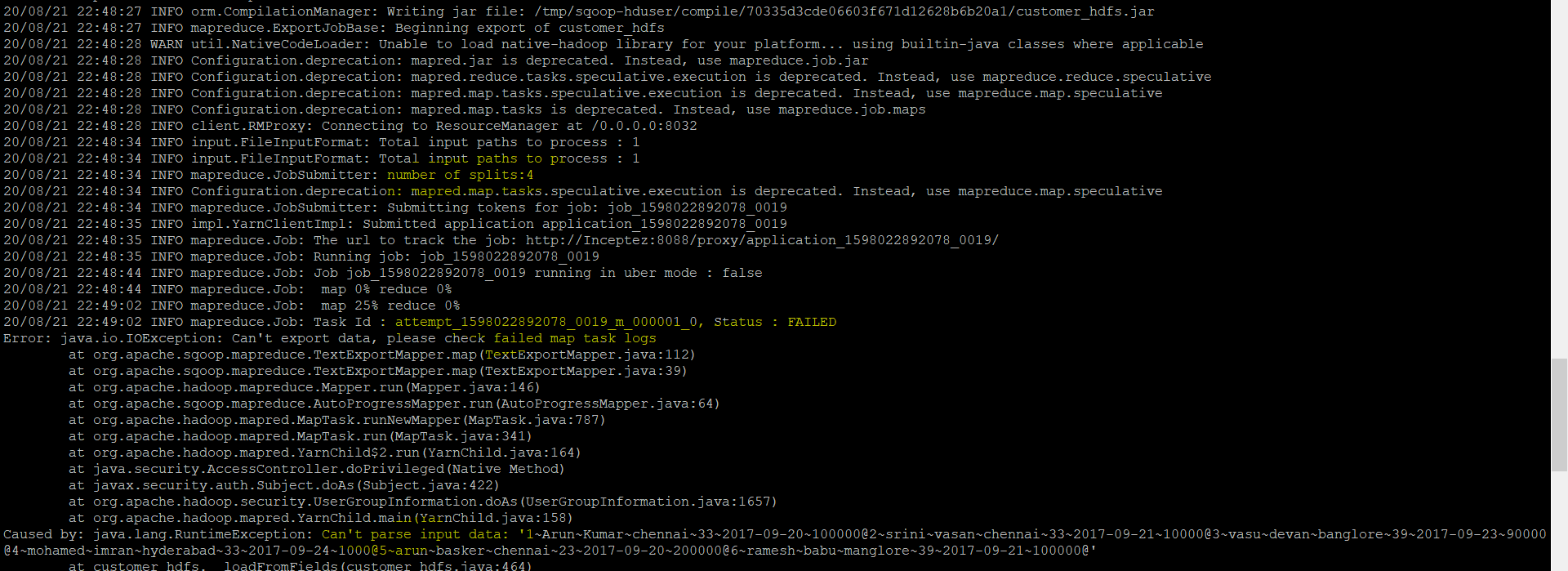
### Export Fail:

sqoop export --connect jdbc:mysql://localhost/custdb --username root --password root --table customer\_hdfs -- export-dir imp\_del;

##### imp\_del:



export fails because of input data and mismatching java target class datatype



sqoop export --connect jdbc:mysql://localhost/custdb --username root --password root --table customer\_hdfs --export-dir imp\_del --fields-terminated-by '~' --lines-terminated-by '\n'

Works – because of explaining how the the columns and rows are separated

sqoop job --create myjob1 -- import --connect jdbc:mysql://localhost/custdb --username root --password root -- table customer --target-dir savedjob --delete-target-dir -m 1

Sqoop –update-mode

Existing sql table

+--------+-------+------+

| rollno | name | age |

+--------+-------+------+

| 18 | basha | 45 |

| 1 | basha | 45 |

| 13 | basha | 45 |

| 7 | basha | 45 |

| 19 | basha | 45 |

| 2 | basha | 45 |

| 14 | basha | 45 |

| 8 | basha | 45 |

| 20 | basha | 45 |

| 3 | basha | 45 |

| 15 | basha | 45 |

| 21 | basha | 45 |

| 9 | basha | 45 |

| 4 | basha | 45 |

| 22 | basha | 45 |

| 5 | basha | 45 |

| 10 | basha | 45 |

| 11 | basha | 45 |

| 6 | basha | 45 |

| 16 | basha | 45 |

| 12 | basha | 45 |

| 17 | basha | 45 |

+--------+-------+------+

New file moved to HDFS

17,nasreen,89

57,uyru,79

1. Sqoop –update-mode -**updateonly** 🡪 will overwrite the rollno =17, name to “nasreen,89”, but will not insert 57.

[hduser@Inceptez ~]$ sqoop export -Dsqoop.export.statements.per.transaction=10 --connect jdbc:mysql://localhost/chkpk --username root --password root --table table1 --export-dir chkpk --batch --update-key rollno --update-mode **updateonly**;

+--------+---------+------+

| rollno | name | age |

+--------+---------+------+

| 18 | basha | 45 |

| 1 | basha | 45 |

| 13 | basha | 45 |

| 7 | basha | 45 |

| 19 | basha | 45 |

| 2 | basha | 45 |

| 14 | basha | 45 |

| 8 | basha | 45 |

| 20 | basha | 45 |

| 3 | basha | 45 |

| 15 | basha | 45 |

| 21 | basha | 45 |

| 9 | basha | 45 |

| 4 | basha | 45 |

| 22 | basha | 45 |

| 5 | basha | 45 |

| 10 | basha | 45 |

| 11 | basha | 45 |

| 6 | basha | 45 |

| 16 | basha | 45 |

| 12 | basha | 45 |

| 17 | nasreen | 89 |

+--------+---------+------+

1. Sqoop –update-mode –**allowinsert** 🡪 will overwrite the rollno =17, name to “nasreen,89”, also insert 57.

mysql> update table1 set name='basha' where rollno=17;

Query OK, 0 rows affected (0.00 sec)

Rows matched: 1 Changed: 0 Warnings: 0

sqoop export -Dsqoop.export.statements.per.transaction=10 --connect jdbc:mysql://localhost/chkpk --username root --password root --table table1 --export-dir chkpk --batch --update-key rollno --update-mode **allowinsert**;

| 57 | uyru | 79 |

| 17 | nasreen | 89 |

+--------+---------+------+

Rollno = 57 is inserted and name is updated with nasreen in rollno 17

When there is allow Update without a PRIMARY-KEY 🡪 Duplicates are created

--update-key custid --update-mode allowinsert

### Sqoop Eval

The eval tool allows users to quickly run simple SQL queries against a database; results are printed to the console. This allows users to preview their import queries to ensure they import the data they expect

sqoop eval \

--connect jdbc:mysql://localhost/custdb \

--username root --password root \

--query "select count(\*) from customer"

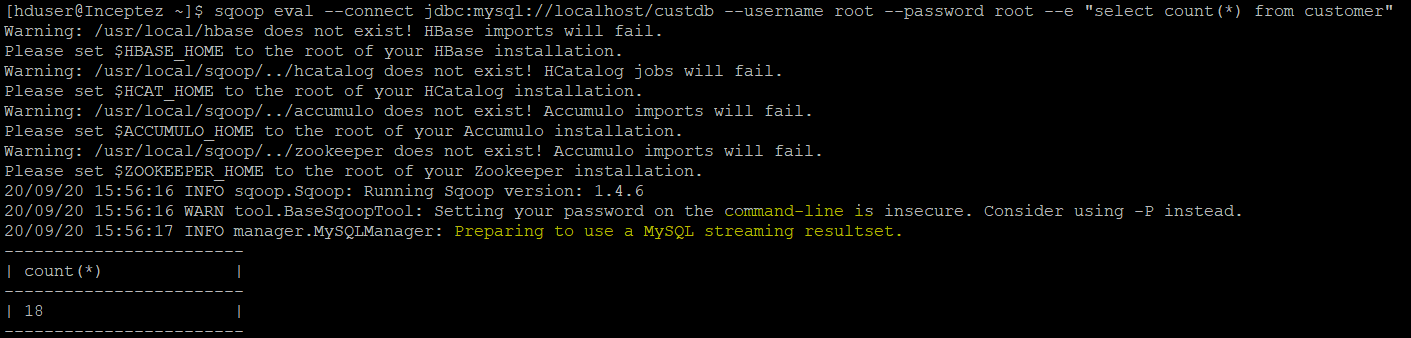
Or

sqoop eval \

--connect jdbc:mysql://localhost/custdb \

--username root --password root \

--e "select count(\*) from customer"



### Sqoop database import:

Import All tables from a DB :

sqoop import-all-tables --connect jdbc:mysql://localhost/custdb \

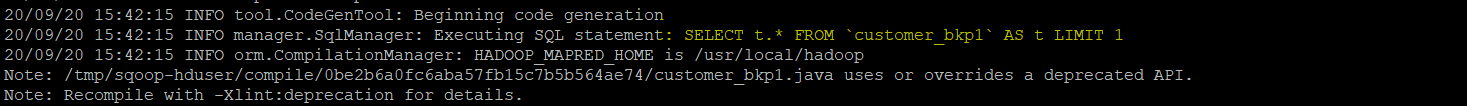
--username root --password root \

--warehousedir '/user/hduser/warehouse/custdb’ \

--m 1 \

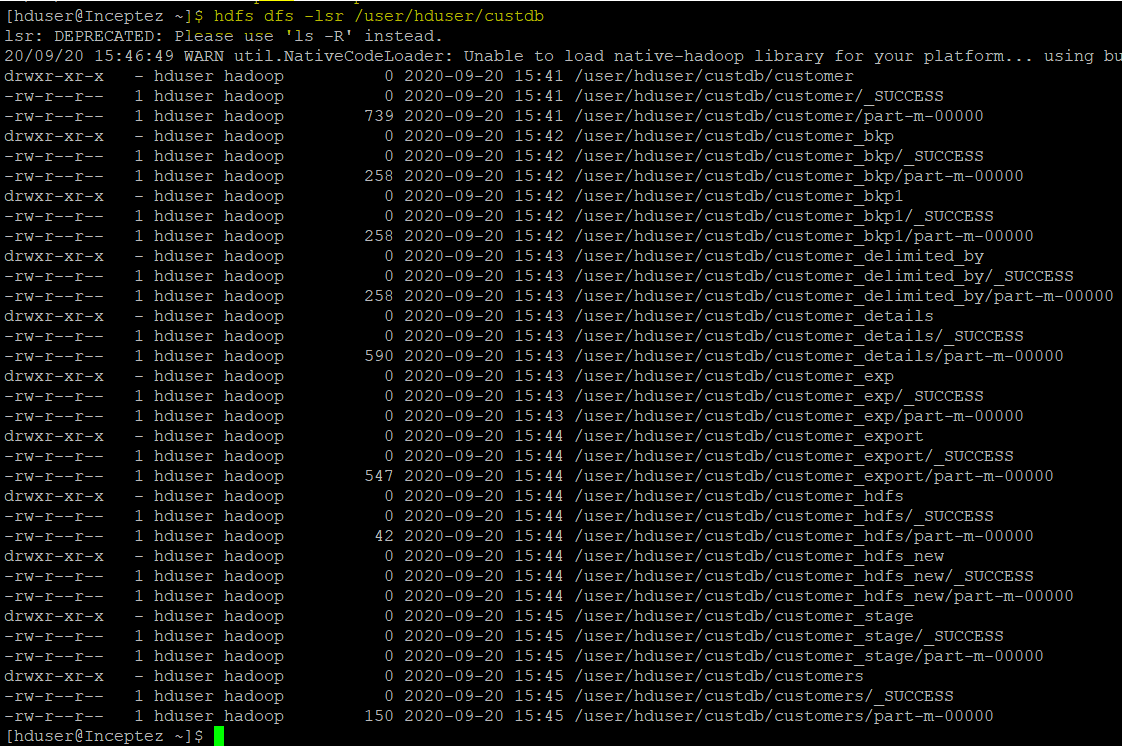
--direct

--Warehouse directory 🡪 Specifies the parent folder where many related files should be stored





All the databases are downloaded in the warehouse directory



### Sqoop import with excluding tables:

sqoop import-all-tables --connect jdbc:mysql://localhost/custdb \

--username root --password root \

--warehouse-dir /user/hduser/warehouse/custdb \

--exclude-tables customer\_bkp,customer\_bkp1,customer\_exp,customer\_delimited\_by,customer\_details \

--m 1 \

--direct \

Only selected tables are exported to hdfs

drwxr-xr-x - hduser hadoop 0 2020-09-20 17:26 /user/hduser/warehouse/custdb/customer

drwxr-xr-x - hduser hadoop 0 2020-09-20 17:26 /user/hduser/warehouse/custdb/customer\_export

drwxr-xr-x - hduser hadoop 0 2020-09-20 17:27 /user/hduser/warehouse/custdb/customer\_hdfs

drwxr-xr-x - hduser hadoop 0 2020-09-20 17:27 /user/hduser/warehouse/custdb/customer\_hdfs\_new

drwxr-xr-x - hduser hadoop 0 2020-09-20 17:28 /user/hduser/warehouse/custdb/customer\_stage

drwxr-xr-x - hduser hadoop 0 2020-09-20 17:28 /user/hduser/warehouse/custdb/customers

when we have –m more than 1 for eg –m 2, we need to specify –split-by or tables should have primary key, other wise error is given

20/09/20 17:33:04 ERROR tool.ImportAllTablesTool: Error during import: No primary key could be found for table customer. Please specify one with --split-by or perform a sequential import wi

th '-m 1'.

### Sqoop Boundary Values:

Sqoop always runs internal auto generated query to estimate the map

sqoop import --connect jdbc:mysql://localhost/custdb \

--username root -password root \

--boundary-query "select min(custid), max(custid) from customers" \

--query 'Select a.custid master\_custid,a.firstname,a.age,a.city,b.custid detail\_custid,a.createdt,b.fulladdress,category from customers a join customer\_details b on a.custid=b.custid WHERE $CONDITIONS' --split-by a.custid \

--split-by a.custid \

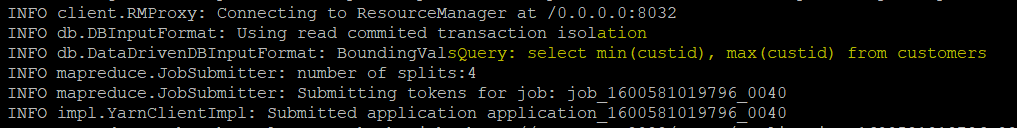
--target-dir ‘/user/hduser/sqoop/boundary\_query’ \

--delete-target-dir \

--m 3 \

--direct

With boundary query



With Auto generated boundary query

sqoop import --connect jdbc:mysql://localhost/custdb \

--username root -password root \

--query 'Select a.custid master\_custid,a.firstname,a.age,a.city,b.custid detail\_custid,a.createdt,b.fulladdress,category from customers a join customer\_details b on a.custid=b.custid WHERE $CONDITIONS' --split-by a.custid \

--split-by a.custid \

--target-dir ‘/user/hduser/sqoop/boundary\_query’ \

--delete-target-dir \

--m 3 \

--direct

Auto generated boundary query:

20/09/20 16:17:40 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(t1.custid), MAX(t1.custid) FROM (Select a.custid master\_custid,a.firstname,a.age,a.city,b.custid detail\_custid,a.createdt,b.fulladdress,category from customers a join customer\_details b on a.custid=b.custid WHERE (1 = 1) ) AS t1

### Batch Export:

sqoop export -Dsqoop.export.statements.per.transaction=10 --connect jdbc:mysql://localhost/chkpk -- username root --password root --table table1 --export-dir tempdata –batch

sqoop import-all-tables --connect jdbc:mysql://localhost/custdb \

--username root --password root \

--query ‘Select a.custid master\_custid,a.firstname,b.custid detail\_custid,a.createdt,a.age,category,transactamt from customers a join customer\_details b on a.custid=b.custid where $CONDITIONS’ \

--boundary-query "select min(custid), max(custid) from customer"\

--split-by a.custid \

--null-string ‘NA’ \

--null-non-string ‘0’ \

--m 3 \

--direct \

--fecth-size 100 \

--compress

### staging-table

Since Sqoop breaks down export process into multiple transactions, it is possible that a failed export job may result in partial data being committed to the database. This can further lead to subsequent jobs failing due to insert collisions in some cases, or lead to duplicated data in others. You can overcome this problem by specifying a staging table via the --staging-table option which acts as an auxiliary table that is used to stage exported data. The staged data is finally moved to the destination table in a single transaction

In order to use the staging facility, you must create the staging table prior to running the export job. This table must be structurally identical to the target table. This table should either be empty before the export job runs, or the --clear-staging-table option must be specified. If the staging table contains data and the --clear-staging-table option is specified, Sqoop will delete all of the data before starting the export job.

### CDC

Change data capture, get only newly inserted row, also called as delta file or increament load

--incremental append --check-column <<col>> --last-value (mostly a date)

### SCD2

Slowly changing dimentions 2

Get newly inserted rows, get modified rows based on check-column

Keeps the history of the data

--increament lastmodified --check-column <<col>> --last-value (anything -date or col)

### SCD1

Slowly changing dimentions 1

Get newly inserted rows, get modified rows based on check-column

Merge the data with original data, delete the old one. Maintain no history

--increament lastmodified --check-column <<col>> --last-value (anything -date or col) --merge-key <<mostly a PK>>

\*\*\* this invokes a reducer, since the shuffling should happen