INSTANT

Short | Fast | Focused

# Apache Sqoop

Transfer data efficiently between RDBMS and the Hadoop ecosystem using the robust Apache Sqoop

Ankit Jain

[PACKT] PUBLISHING

# Instant Apache Sqoop

Transfer data efficiently between RDBMS and the Hadoop ecosystem using the robust Apache Sqoop

**Ankit Jain**

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

# Instant Apache Sqoop

First published: August 2013

Production Reference: 1230813

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-576-7

www.packtpub.com

# Credits

**Author**
Ankit Jain

**Reviewer**
Jagat Jasjit Singh

**Acquisition Editors**
Kartikey Pandey
Ashwin Nair

**Commissioning Editor**
Mohammed Fahad

**Technical Editors**
Vrinda Nitesh Bhosale
Pratik More

**Copy Editors**
Gladson Monteiro
Sayanee Mukherjee

**Project Coordinator**
Romal Karani

**Proofreader**
Joanna McMahon

**Graphics**
Yuvraj Mannari

**Production Coordinator**
Nitesh Thakur

**Cover Work**
Nitesh Thakur

**Cover Image**
Ronak Dhruv

# About the Author

**Ankit Jain** is a software professional with over two years of experience in implementing, designing, and managing Big Data solutions for industry leaders. His core skills include Hadoop, HBase, Hive, Sqoop, Flume, Elasticsearch, Machine Learning, Kafka, Storm, Java, and J2EE. He is currently employed with Impetus Infotech Pvt Ltd.

He is an active blogger and can be followed at `http://ankitasblogger.blogspot.in/`.

# About the Reviewer

**Jagat Jasjit Singh** was born in the holy land of India. He holds a Bachelor's degree in Electrical Engineering and a Master's degree in IT from Guru Nanak Dev University (Amritsar) with research focus on Accessibility of Internet.

He fell in love with computers when, during his time at college, he founded his own company (MyTutorOnline) offering an online tutoring platform to students and teachers around the world.

In his present job, he architects and designs Big Data and machine learning solutions for various companies.

You can connect with him via LinkedIn at `http://au.linkedin.com/in/jagatsingh`.

You can also comment on his blog at `http://jugnu-life.blogspot.com/`.

I am thankful to Almighty God for all his blessings.

# Table of Contents

# Preface

Welcome to *Instant Apache Sqoop*. While reading this book, you will be taken on a journey to transfer data between Hadoop and databases. We will start with a general introduction to the Apache Sqoop import features, which covers some common arguments required in the Sqoop import statement, and also a few examples to import data from RDBMS to Hadoop. You will also learn how we can append data into an already imported table.

This book will also focus on importing data into HBase and Hive. In addition to all that, you'll see how you can populate the HBase and Hive tables.

This book will also cover some common arguments that are required in Sqoop's export features, and also a few examples to export data back to RDBMS from Hadoop and Hive.

Finally, this book will show you some third-party Sqoop connectors, and how we can install and use these connectors into Sqoop.

## What this book covers

*Working with the import process (Intermediate)* provides a high-level overview of Sqoop's import architecture and import commands to dump the RDBMS data into Hadoop.

*Incremental import (Simple)* explains how to append data, reimport the modified records, and save the Sqoop job.

*Populating the HBase table (Simple)* covers an introduction to HBase, the HBase table structure, and the HBase basic commands.

*Importing data into HBase (Intermediate)* explains the arguments that are required in a Sqoop statement to load data into HBase. It also covers sample examples to transfer data from RDBMS to HBase.

*Populating the Hive table (Simple)* covers an introduction of Hive and its basic commands.

*Importing data into Hive (Simple)* explains arguments required in a Sqoop statement to load data into Hive. It also covers sample examples to transfer data from RDBMS to Hive.

*The exporting process (Intermediate)* provides a high-level overview of Sqoop's export architecture and export commands to transfer process data back to RDBMS from Hadoop.

*Exporting data from Hive (Simple)* covers commands to export data from the Hive table to RDBMS.

*Using Sqoop connectors (Advanced)* explains some third-party Sqoop connectors available in the market.

## What you need for this book

This book was written using Apache Sqoop 1.4.3, and all the examples and functions should work with it.

In addition to that, some chapters require additional software, such as Hadoop (Version 1.0.4), MySQL, HBase (Version 0.95.2), JDK 1.6 and Hive (Version 0.7), but when needed, this is explicitly mentioned.

## Who this book is for

This book will guide you through the basics of Apache Sqoop, and how we can effectively and efficiently move data between Hadoop and RDBMS.

If you know Apache Sqoop and have worked with it, you may find this book interesting as it provides a good overview of all the functionalities with examples and descriptions. This book covers all the important features of Sqoop.

This book also gives an overview of third-party Sqoop connectors and sample import/export examples of each connector.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:
"We will call the Sqoop installation directory as `$SQOOP_HOME`."

Any command-line input or output is written as follows:

```
$ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1 --username
root --password password --table tableName --target-dir /user/abc/
tableName
```

[ Warnings or important notes appear in a box like this. ]

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

### Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

### Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

# Instant Apache Sqoop

Welcome to *Instant Apache Sqoop*. This book contains an introduction to Apache Sqoop's import and export features, which will cover the arguments required in Sqoop's import and export processes. We will also learn how to append data into an existing table. At the end, this book will show you some third-party Sqoop connectors, and how we can install and use these connectors into Sqoop.

## Working with the import process (Intermediate)

Sqoop is an Apache Hadoop top-level project and designed to move data between Hadoop and RDBMS. Sqoop has two important features:

- Import
- Export

Sqoop import means loading traditional RDBMS data into Hadoop, HBase, and Hive. The Sqoop import process will take the database table as the input, read the table row-by-row, and produce a list of files as the output into Hadoop. The output files contain a copy of the imported table. The output of the import process is a set of files because the import process is performed in parallel.

This recipe will cover common arguments of the import process and also cover sample examples of the import process.

## Getting ready

The first thing we need to do is to download the latest version of Sqoop from `http://www.apache.org/dist/sqoop/` and extract it on your machine. We will call the Sqoop installation directory as `$SQOOP_HOME`.

Given here are the prerequisites for the Sqoop import process:

- Installed and running Relational Database Management System (MySQL)
- Installed and running Hadoop cluster
- Set `$HADOOP_HOME` environment variable

The following table shows the common arguments of the import process:

| **Parameters** | **Description** |
|---|---|
| `--connect <jdbc-uri>` | Specifies the server or database to connect to. It also specifies the port. For example: `--connect jdbc:mysql://host:port/databaseName` |
| `--connection-manager <class-name>` | Specifies the connection manager class name. |
| `--driver <class-name>` | Specifies the fully qualified name of the JDBC driver class. |
| `--hadoop-home <dir>` | This parameter is used to override the `$HADOOP_HOME` environment variable. |
| `-P` | If a user doesn't want to specify the database password along with the command, we can use the `–P` option to read the password from the console. |
| `--password <password>` | Sets the authentication password required to connect to the input source. |
| `--username <username>` | Sets the authentication username. |
| `--connection-param-file <properties-file>` | Specifies the connection parameter's file. |
| `--help` | This option will provide the usage instructions. |
| `--verbose` | Prints more information during a query execution. |

## How to do it...

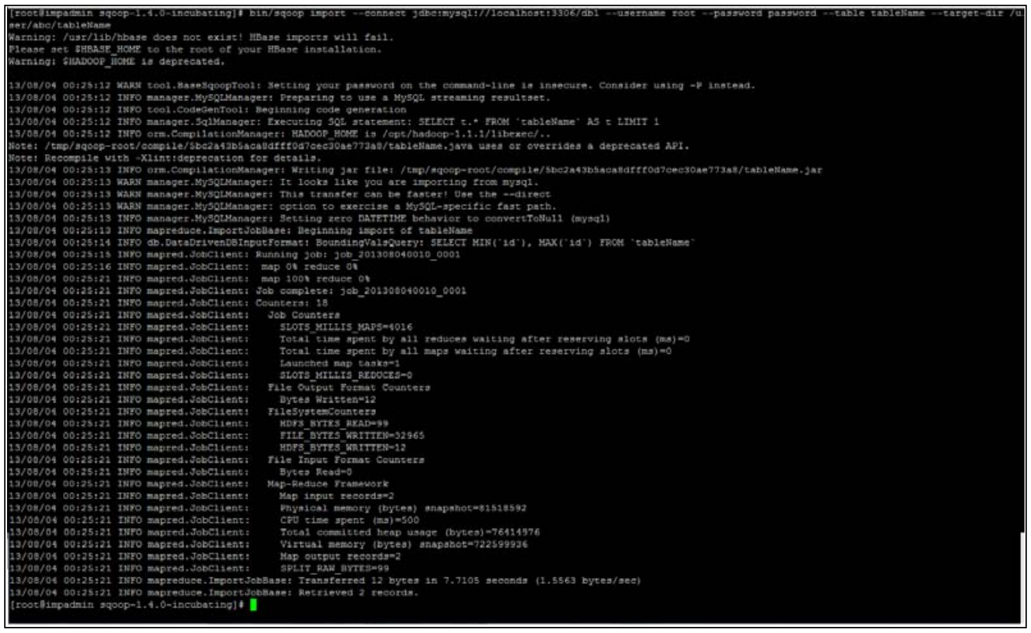
Let's see how to work with the import process:

1. First, we will go through the architecture of the import process.
2. Then, we will start with importing a single RDBMS table into Hadoop.

   **Query 1**:

   ```
   $ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1
   --username root --password password --table tableName --target-dir
   /user/abc/tableName
   ```

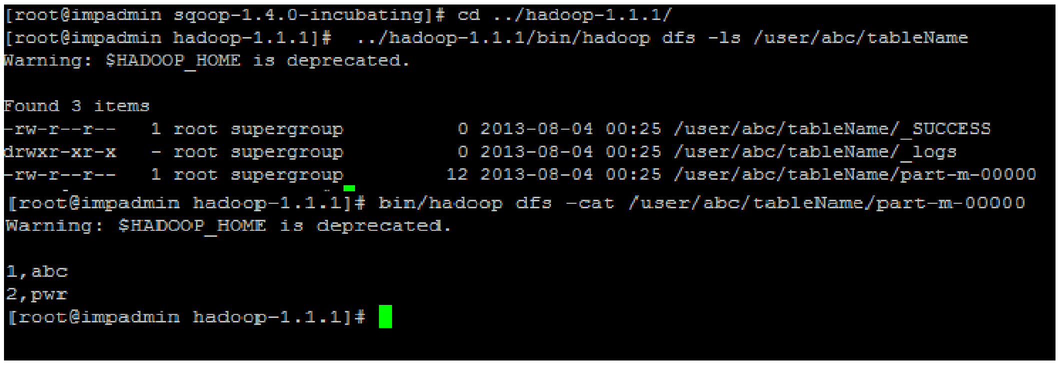
   Run the query as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop import --connect jdbc:mysql://localhost:3306/db1 --username root --password password --table tableName --target-dir /u
ser/abc/tableName
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: $HADOOP_HOME is deprecated.

13/08/04 00:25:12 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 00:25:12 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 00:25:12 INFO tool.CodeGenTool: Beginning code generation
13/08/04 00:25:12 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 00:25:12 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/5bc2a43b5aca8dfff0d7cec30ae773a8/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 00:25:13 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/5bc2a43b5aca8dfff0d7cec30ae773a8/tableName.jar
13/08/04 00:25:13 WARN manager.MySQLManager: It looks like you are importing from mysql.
13/08/04 00:25:13 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
13/08/04 00:25:13 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
13/08/04 00:25:13 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
13/08/04 00:25:13 INFO mapreduce.ImportJobBase: Beginning import of tableName
13/08/04 00:25:14 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`id`), MAX(`id`) FROM `tableName`
13/08/04 00:25:15 INFO mapred.JobClient: Running job: job_201308040010_0001
13/08/04 00:25:16 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 00:25:21 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 00:25:21 INFO mapred.JobClient: Job complete: job_201308040010_0001
13/08/04 00:25:21 INFO mapred.JobClient: Counters: 18
13/08/04 00:25:21 INFO mapred.JobClient:   Job Counters
13/08/04 00:25:21 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=4016
13/08/04 00:25:21 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 00:25:21 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 00:25:21 INFO mapred.JobClient:     Launched map tasks=1
13/08/04 00:25:21 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 00:25:21 INFO mapred.JobClient:   File Output Format Counters
13/08/04 00:25:21 INFO mapred.JobClient:     Bytes Written=12
13/08/04 00:25:21 INFO mapred.JobClient:   FileSystemCounters
13/08/04 00:25:21 INFO mapred.JobClient:     HDFS_BYTES_READ=99
13/08/04 00:25:21 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=32965
13/08/04 00:25:21 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=12
13/08/04 00:25:21 INFO mapred.JobClient:   File Input Format Counters
13/08/04 00:25:21 INFO mapred.JobClient:     Bytes Read=0
13/08/04 00:25:21 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 00:25:21 INFO mapred.JobClient:     Map input records=2
13/08/04 00:25:21 INFO mapred.JobClient:     Physical memory (bytes) snapshot=81518592
13/08/04 00:25:21 INFO mapred.JobClient:     CPU time spent (ms)=500
13/08/04 00:25:21 INFO mapred.JobClient:     Total committed heap usage (bytes)=76414976
13/08/04 00:25:21 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=722599936
13/08/04 00:25:21 INFO mapred.JobClient:     Map output records=2
13/08/04 00:25:21 INFO mapred.JobClient:     SPLIT_RAW_BYTES=99
13/08/04 00:25:21 INFO mapreduce.ImportJobBase: Transferred 12 bytes in 7.7105 seconds (1.5563 bytes/sec)
13/08/04 00:25:21 INFO mapreduce.ImportJobBase: Retrieved 2 records.
[root@impadmin sqoop-1.4.0-incubating]#
```

The content of the output file in HDFS will look like the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# cd ../hadoop-1.1.1/
[root@impadmin hadoop-1.1.1]#  ../hadoop-1.1.1/bin/hadoop dfs -ls /user/abc/tableName
Warning: $HADOOP_HOME is deprecated.

Found 3 items
-rw-r--r--   1 root supergroup          0 2013-08-04 00:25 /user/abc/tableName/_SUCCESS
drwxr-xr-x   - root supergroup          0 2013-08-04 00:25 /user/abc/tableName/_logs
-rw-r--r--   1 root supergroup         12 2013-08-04 00:25 /user/abc/tableName/part-m-00000
[root@impadmin hadoop-1.1.1]# bin/hadoop dfs -cat /user/abc/tableName/part-m-00000
Warning: $HADOOP_HOME is deprecated.

1,abc
2,pwr
[root@impadmin hadoop-1.1.1]#
```

**Query 2**:

```
$ bin/sqoop import --options-file /opt/option-file.txt --table
tableName --target-dir /user/abc/tableName
```

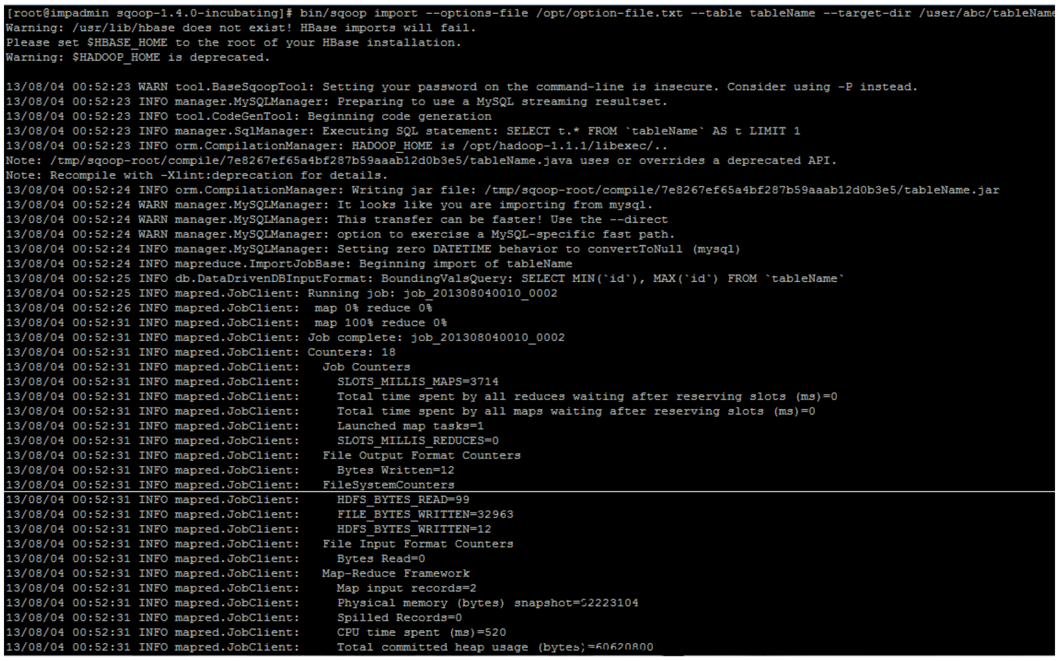
Here, the options file (`/opt/option-file.txt`) contains the following lines:

```
--connect
jdbc:mysql://localhost/db1
--username
root
--password
password
```

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Run the query as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop import --options-file /opt/option-file.txt --table tableName --target-dir /user/abc/tableName
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: $HADOOP_HOME is deprecated.

13/08/04 00:52:23 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 00:52:23 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 00:52:23 INFO tool.CodeGenTool: Beginning code generation
13/08/04 00:52:23 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 00:52:23 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/7e8267ef65a4bf287b59aaab12d0b3e5/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 00:52:24 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/7e8267ef65a4bf287b59aaab12d0b3e5/tableName.jar
13/08/04 00:52:24 WARN manager.MySQLManager: It looks like you are importing from mysql.
13/08/04 00:52:24 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
13/08/04 00:52:24 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
13/08/04 00:52:24 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
13/08/04 00:52:24 INFO mapreduce.ImportJobBase: Beginning import of tableName
13/08/04 00:52:25 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`id`), MAX(`id`) FROM `tableName`
13/08/04 00:52:25 INFO mapred.JobClient: Running job: job_201308040010_0002
13/08/04 00:52:26 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 00:52:31 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 00:52:31 INFO mapred.JobClient: Job complete: job_201308040010_0002
13/08/04 00:52:31 INFO mapred.JobClient: Counters: 18
13/08/04 00:52:31 INFO mapred.JobClient:   Job Counters
13/08/04 00:52:31 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=3714
13/08/04 00:52:31 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 00:52:31 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 00:52:31 INFO mapred.JobClient:     Launched map tasks=1
13/08/04 00:52:31 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 00:52:31 INFO mapred.JobClient:   File Output Format Counters
13/08/04 00:52:31 INFO mapred.JobClient:     Bytes Written=12
13/08/04 00:52:31 INFO mapred.JobClient:   FileSystemCounters
13/08/04 00:52:31 INFO mapred.JobClient:     HDFS_BYTES_READ=99
13/08/04 00:52:31 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=32963
13/08/04 00:52:31 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=12
13/08/04 00:52:31 INFO mapred.JobClient:   File Input Format Counters
13/08/04 00:52:31 INFO mapred.JobClient:     Bytes Read=0
13/08/04 00:52:31 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 00:52:31 INFO mapred.JobClient:     Map input records=2
13/08/04 00:52:31 INFO mapred.JobClient:     Physical memory (bytes) snapshot=52223104
13/08/04 00:52:31 INFO mapred.JobClient:     Spilled Records=0
13/08/04 00:52:31 INFO mapred.JobClient:     CPU time spent (ms)=520
13/08/04 00:52:31 INFO mapred.JobClient:     Total committed heap usage (bytes)=60620800
```

3. Then we will cover how we can change the default field and the line delimiter.

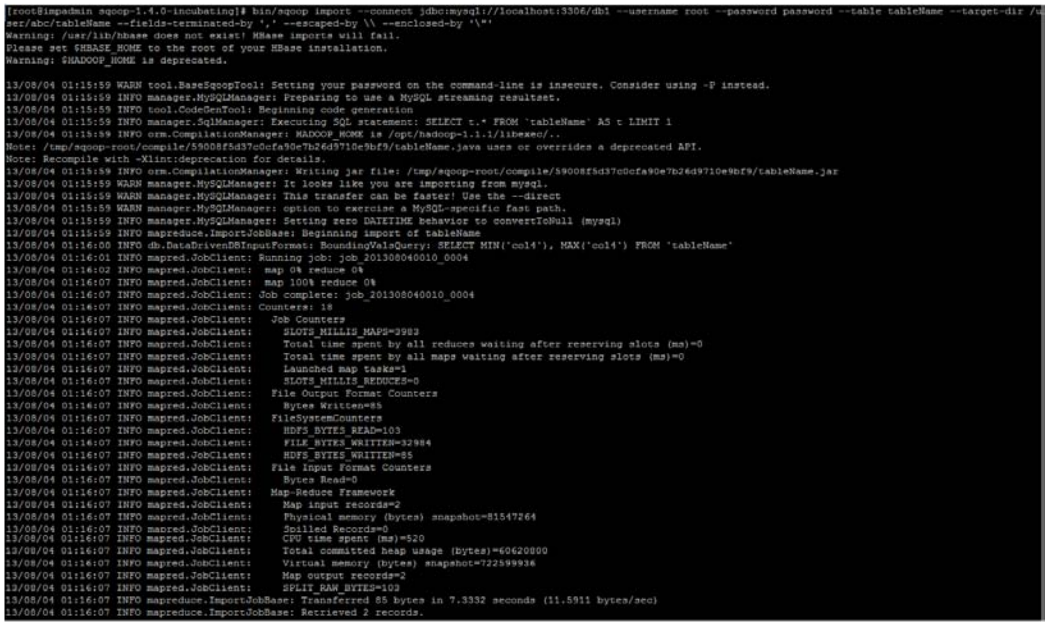
   Let's consider the input table that contains four columns: `Col1`, `Col2`, `Col3`, and `Col4`.

| **Col4** | **Col1** | **Col2** | **Col3** |
|---|---|---|---|
| 1 | `Hi All, I am good` | `ABC` | `XYZ` |
| 2 | `My Book name is "Apache Sqoop"` | `ASD` | `ZXC` |

**Query 3**:

```
$ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1
--username root --password password --table tableName --target-
dir /user/abc/tableName --fields-terminated-by ',' --escaped-by \\
--enclosed-by '\"'
```

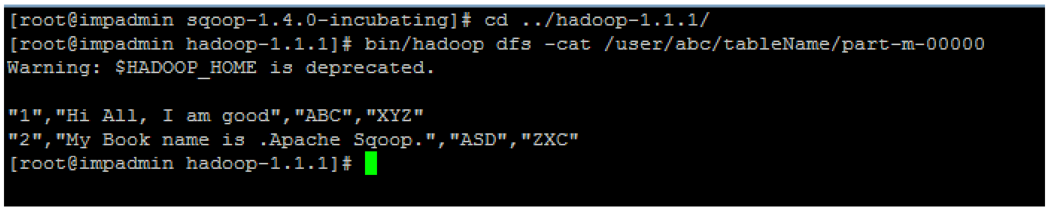
Run the query as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop import --connect jdbc:mysql://localhost:3306/db1 --username root --password password --table tableName --target-dir /u
ser/abc/tableName --fields-terminated-by ',' --escaped-by \\ --enclosed-by '\"'
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: $HADOOP_HOME is deprecated.

13/08/04 01:15:59 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 01:15:59 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 01:15:59 INFO tool.CodeGenTool: Beginning code generation
13/08/04 01:15:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 01:15:59 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/59008f5d37c0cfa90e7b26d9710e9bf9/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 01:15:59 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/59008f5d37c0cfa90e7b26d9710e9bf9/tableName.jar
13/08/04 01:15:59 WARN manager.MySQLManager: It looks like you are importing from mysql.
13/08/04 01:15:59 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
13/08/04 01:15:59 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
13/08/04 01:15:59 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
13/08/04 01:15:59 INFO mapreduce.ImportJobBase: Beginning import of tableName
13/08/04 01:16:00 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN('col4'), MAX('col4') FROM 'tableName'
13/08/04 01:16:01 INFO mapred.JobClient: Running job: job_201308040010_0004
13/08/04 01:16:02 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 01:16:07 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 01:16:07 INFO mapred.JobClient: Job complete: job_201308040010_0004
13/08/04 01:16:07 INFO mapred.JobClient: Counters: 18
13/08/04 01:16:07 INFO mapred.JobClient:   Job Counters
13/08/04 01:16:07 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=3983
13/08/04 01:16:07 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 01:16:07 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 01:16:07 INFO mapred.JobClient:     Launched map tasks=1
13/08/04 01:16:07 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 01:16:07 INFO mapred.JobClient:   File Output Format Counters
13/08/04 01:16:07 INFO mapred.JobClient:     Bytes Written=85
13/08/04 01:16:07 INFO mapred.JobClient:   FileSystemCounters
13/08/04 01:16:07 INFO mapred.JobClient:     HDFS_BYTES_READ=103
13/08/04 01:16:07 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=32984
13/08/04 01:16:07 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=85
13/08/04 01:16:07 INFO mapred.JobClient:   File Input Format Counters
13/08/04 01:16:07 INFO mapred.JobClient:     Bytes Read=0
13/08/04 01:16:07 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 01:16:07 INFO mapred.JobClient:     Map input records=2
13/08/04 01:16:07 INFO mapred.JobClient:     Physical memory (bytes) snapshot=81547264
13/08/04 01:16:07 INFO mapred.JobClient:     Spilled Records=0
13/08/04 01:16:07 INFO mapred.JobClient:     CPU time spent (ms)=520
13/08/04 01:16:07 INFO mapred.JobClient:     Total committed heap usage (bytes)=60620800
13/08/04 01:16:07 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=722599936
13/08/04 01:16:07 INFO mapred.JobClient:     Map output records=2
13/08/04 01:16:07 INFO mapred.JobClient:     SPLIT_RAW_BYTES=103
13/08/04 01:16:07 INFO mapreduce.ImportJobBase: Transferred 85 bytes in 7.3332 seconds (11.5911 bytes/sec)
13/08/04 01:16:07 INFO mapreduce.ImportJobBase: Retrieved 2 records.
```

The content of the output file in HDFS will look like the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# cd ../hadoop-1.1.1/
[root@impadmin hadoop-1.1.1]# bin/hadoop dfs -cat /user/abc/tableName/part-m-00000
Warning: $HADOOP_HOME is deprecated.

"1","Hi All, I am good","ABC","XYZ"
"2","My Book name is .Apache Sqoop.","ASD","ZXC"
[root@impadmin hadoop-1.1.1]#
```

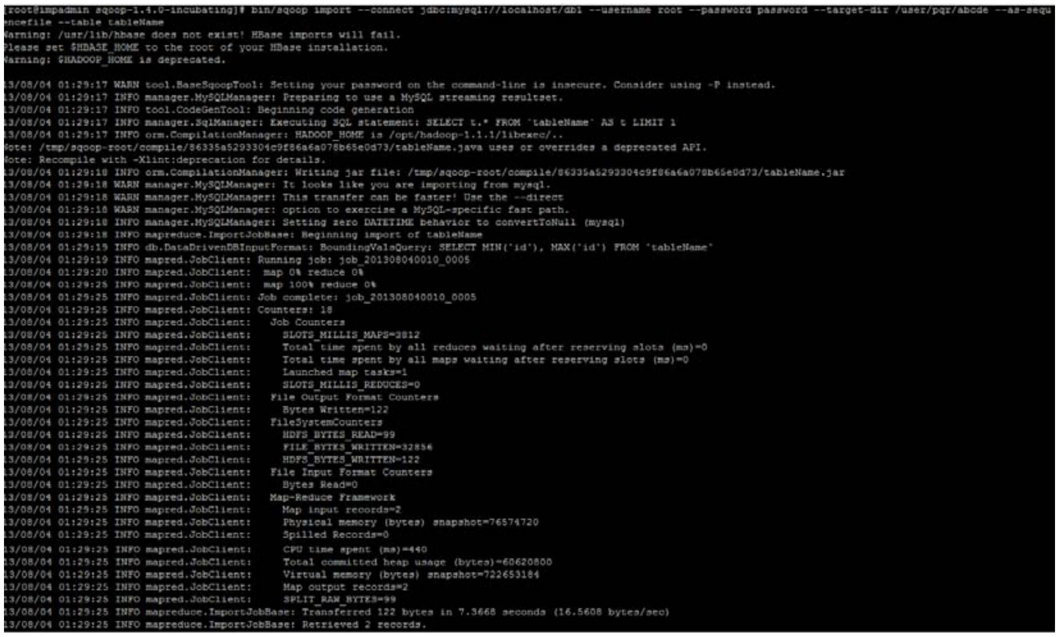
4. Now we will go through the default output format of the import process, and how we can modify the default format.

   The following command is used to store data in a SequenceFile format:

   **Query 4**:

   ```
   $ bin/sqoop import --connect jdbc:mysql://localhost/db1 --username
   root --password password --target-dir /user/pqr/abcde --as-
   sequencefile --table tableName
   ```

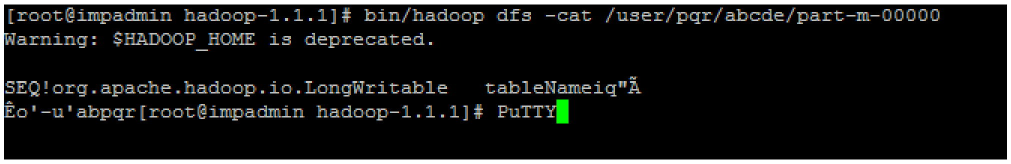
   We will run the command as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop import --connect jdbc:mysql://localhost/db1 --username root --password password --target-dir /user/pqr/abcde --as-sequ
encefile --table tableName
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: $HADOOP_HOME is deprecated.

13/08/04 01:29:17 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 01:29:17 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 01:29:17 INFO tool.CodeGenTool: Beginning code generation
13/08/04 01:29:17 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 01:29:17 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/86335a5293304c9f86a6a078b65e0d73/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 01:29:18 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/86335a5293304c9f86a6a078b65e0d73/tableName.jar
13/08/04 01:29:18 WARN manager.MySQLManager: It looks like you are importing from mysql.
13/08/04 01:29:18 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
13/08/04 01:29:18 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
13/08/04 01:29:18 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
13/08/04 01:29:18 INFO mapreduce.ImportJobBase: Beginning import of tableName
13/08/04 01:29:19 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`id`), MAX(`id`) FROM `tableName`
13/08/04 01:29:19 INFO mapred.JobClient: Running job: job_201308040010_0005
13/08/04 01:29:20 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 01:29:25 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 01:29:25 INFO mapred.JobClient: Job complete: job_201308040010_0005
13/08/04 01:29:25 INFO mapred.JobClient: Counters: 18
13/08/04 01:29:25 INFO mapred.JobClient:   Job Counters
13/08/04 01:29:25 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=3812
13/08/04 01:29:25 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 01:29:25 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 01:29:25 INFO mapred.JobClient:     Launched map tasks=1
13/08/04 01:29:25 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 01:29:25 INFO mapred.JobClient:   File Output Format Counters
13/08/04 01:29:25 INFO mapred.JobClient:     Bytes Written=122
13/08/04 01:29:25 INFO mapred.JobClient:   FileSystemCounters
13/08/04 01:29:25 INFO mapred.JobClient:     HDFS_BYTES_READ=99
13/08/04 01:29:25 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=32856
13/08/04 01:29:25 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=122
13/08/04 01:29:25 INFO mapred.JobClient:   File Input Format Counters
13/08/04 01:29:25 INFO mapred.JobClient:     Bytes Read=0
13/08/04 01:29:25 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 01:29:25 INFO mapred.JobClient:     Map input records=2
13/08/04 01:29:25 INFO mapred.JobClient:     Physical memory (bytes) snapshot=76574720
13/08/04 01:29:25 INFO mapred.JobClient:     Spilled Records=0
13/08/04 01:29:25 INFO mapred.JobClient:     CPU time spent (ms)=440
13/08/04 01:29:25 INFO mapred.JobClient:     Total committed heap usage (bytes)=60620800
13/08/04 01:29:25 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=722653184
13/08/04 01:29:25 INFO mapred.JobClient:     Map output records=2
13/08/04 01:29:25 INFO mapred.JobClient:     SPLIT_RAW_BYTES=99
13/08/04 01:29:25 INFO mapreduce.ImportJobBase: Transferred 122 bytes in 7.3668 seconds (16.5608 bytes/sec)
13/08/04 01:29:25 INFO mapreduce.ImportJobBase: Retrieved 2 records.
```

The content of the output file in HDFS will look like the following screenshot:



```
[root@impadmin hadoop-1.1.1]# bin/hadoop dfs -cat /user/pqr/abcde/part-m-00000
Warning: $HADOOP_HOME is deprecated.

SEQ!org.apache.hadoop.io.LongWritable   tableNameiq"Ã
Êo'-u'abpqr[root@impadmin hadoop-1.1.1]# PuTTY
```

5. Next we will go through the direct access mode.

   **Query 5**:

   ```
   $ bin/sqoop import --connect jdbc:mysql://localhost/db1 --username
   root --password password --target-dir /user/pqr/abcde --direct
   --table tableName
   ```

6. Finally we will put some light on the approach of the import-only selected rows and columns of the RDBMS table into Hadoop.

   **Query 6**: Importing the selected columns.

   ```
   $ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1
   --username root --password password --table student --target-dir /
   user/abc/student --columns "student_id,address,name"
   ```

   **Query 7**: Importing the selected rows.

   ```
   $ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1
   --username root --password password --table student --target-dir /
   user/abc/student --where 'student_id<100'
   ```

   **Query 8**: Importing the selected columns of the selected rows.

   ```
   $ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1
   --username root --password password --table student --target-dir
   /user/abc/student --columns "student_id,address,name" -- where
   'student_id<100'
   ```

## How it works...

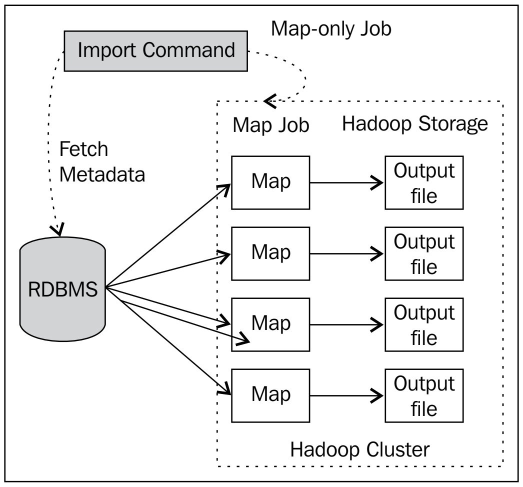
Now let's see how the preceding steps work.

### Architecture of the import process

Let's look at the architecture of the Sqoop import process. The end-to-end process as depicted in the following diagram is broken down into two steps:

1. Sqoop first connects to the database server to pull the desired metadata information from the input table.
2. In the second step, Sqoop executes a MapReduce job on the Hadoop cluster. MapReduce will use the metadata generated in step 1 to perform the actual import process.

The output of the process, depicted in the following diagram, is a list of files. By default, the output files generated in HDFS contain comma-delimited fields, while lines (records) are separated by new lines. Users can overwrite the default field delimiters and line terminators by explicitly specifying the field separator and line terminator characters in the Sqoop import statement.



## Importing a single table

Apart from the common arguments of the import process, as explained previously, this part covers some other arguments that are required to import a table into the Hadoop Distributed File System.

| Parameters | Description |
|---|---|
| `--table <table-name>` | Name of the input table to fetch. |
| `--target-dir<dir>` | Location of the output/target directory in HDFS. |
| `--direct` | This parameter is used if a user wants to use a non-JDBC-based access mechanism for faster database access. |
| `--options-file <file-path>` | All the command-line options that are common in most of the commands can be put in the options file for convenience. |

Query 1 will run a MapReduce job and import all the rows of the given table to HDFS where `/user/abc/tableName` is the location of the output files. The records imported in HDFS preserve their original columns' order, which means that if an input table contains four columns A, B, C, and D, the content in the HDFS file will look like this:

```
A1, B1, C1, D1
A2, B2, C2, D2
```

As some of the arguments such as `--connect`, `--username`, and `–password` are common in most of the input commands, we can put these arguments into the option file for convenience.

By using the `–option-file` argument, Query 1 can also be written as Query 2.

## Field and line terminator

By default, Sqoop uses a comma (`,`) as the field delimiter and a new line (`\n`) as the row separator/terminator in an output file, but we can overwrite this default delimiter by using the arguments listed in the following table:

| Parameters | Description |
|---|---|
| `--fields-terminated-by <char>` | Specifies the field separator character. |
| `--lines-terminated-by <char>` | Specifies the line terminator or the record terminator character. |

The choice of a delimiter is important in an import process; if the field delimiter is already present in the field value, it will create unambiguous parsing of the data during the analysis phase.

For example, if the input value of any column is, for example, `31, ABC`, we should not be importing data by using a comma as a field delimiter; otherwise, it will create unambiguous parsing of the data during the analysis phase.

We can remove ambiguity while parsing the data by using the arguments, shown in the following table, in Sqoop statements:

| Parameters | Description |
|---|---|
| `--enclosed-by <char>` | Specifies the field-enclosed character. |
| `--escaped-by <char>` | Specifies the escaped character. |

Query 3 shows the use of `–field-terminated-by`, `--enclosed-by`, and `--escaped-by` arguments.

## Supported output format

The delimited text is the default import format, but we can import data into binary Avro or SequenceFile format. If the user wants to analyze data using MapReduce, it is always recommended to use the SequenceFile format because reading from a SequenceFile is much faster than reading from a delimited text file.

| Parameters | Description |
|---|---|
| `--as-avrodatafile` | Storing data in an Avro file. |
| `--as-sequencefile` | Storing data in a SequenceFile format. |
| `--as-textfile` | Importing data as a text file. |

Query 4 covered an example on using `–as-sequencefile` as the output format.

## Direct access mode

As we know, some databases support non-JDBC-based access mechanisms to provide faster database access. Sqoop also supports the non-JDBC access mechanism for the following databases:

- MySQL (5.0+)
- PostgreSQL (8.3+)—only for import process

Query 5 shows the use of MySQL direct mode.

## Importing selected columns

By default, the import query will select all the columns of the input table for import, but we can select the subset of columns by specifying the comma-separated list of columns in the `--columns` argument.

Query 6 will only fetch three columns (`student_id`, `address`, and `name`) of the `student` table. If the import query contains the `--columns` argument, the order of the columns in the output file is the same as the order specified in the `--columns` argument. The output in HDFS will look like this:

```
student_id, address, name
1, Delhi, XYZ
2, Mumbai, PQR
..........
```

If the input query contains the column in an order that starts with `address`, then `name`, and then `student_id`, the output in HDFS will look like this:

```
address, name, student_id
Delhi, XYZ, 1
Mumbai, PQR, 2
.............
```

### Importing selected rows

By default, all the rows of the input table will be imported to HDFS, but we can control which rows need to be imported by using a `--where` argument in the import statement.

Query 7 will import only those rows into HDFS where the value of the `student_id` column is greater than 100.

Query 8 uses both `--columns` and `--where` arguments in one statement.

For Query 8, Sqoop will internally generate the query of the following form:

```
select student_id, address, name from student where student_id<100
```

## There's more...

This section covers some other arguments that we can use to make the import process more effective and efficient.

### Free form query imports

The preceding sections explain the usage of the `--table`, `--columns`, and `--where` arguments. This section introduces the `--query` argument. We can specify any valid SQL statement as a value of `--query` arguments.

**Query 9**:

```
$ bin/sqoop import --connect jdbc:mysql://localhost:3306/db1 --username
root --password password --query "select student_id, name, addess from
student where student_id< 100 and $CONDITIONS" --target-dir /user/abc/
student -split-by id
```

The preceding query will be similar to Query 8 in terms of the import operation, that means, we can use an `--query` argument in place of multiple arguments (`--table`, `--columns`, and `--where`).

### Importing all tables

So far we have imported a single table into HDFS. This section introduces an `import-all-tables` tool by which we can import a set of tables from an RDBMS to HDFS. The `import-all-tables` tool creates a separate directory in HDFS for each RDBMS table. The following are the mandatory conditions for the `import-all-tables` tool:

- All tables must have a single primary key column
- A user must intend to import all the columns of each table
- No `--where`, `--columns`, and `--query` arguments are permitted

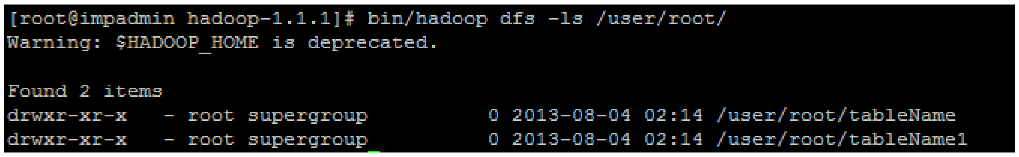
Consider the following example:

**Query 10**:

```
$ bin/sqoop import-all-tables --connect jdbc:mysql://localhost:3306/db1
--username root --password password
```

This query will import all the tables of database `db1` into HDFS.

Output directories in HDFS appear as shown in the following screenshot:



```
[root@impadmin hadoop-1.1.1]# bin/hadoop dfs -ls /user/root/
Warning: $HADOOP_HOME is deprecated.

Found 2 items
drwxr-xr-x   - root supergroup          0 2013-08-04 02:14 /user/root/tableName
drwxr-xr-x   - root supergroup          0 2013-08-04 02:14 /user/root/tableName1
```

## Parallelism arguments

As described earlier, Sqoop runs the MapReduce job to import the RDBMS data into HDFS. This section covers some arguments by which a user can control the number of map tasks that will execute during the import process.

| Parameter | Description |
|---|---|
| `-m , --num-mapper <n>` | We can control the number of map tasks by using the `–m` argument. |
| `--split-by <column-name>` | Specifies the column of the table used to split the work between maps. |

The`–m` argument takes an integer value that corresponds to the number of map tasks that will run. By default, four map tasks will be executed. It always recommends to use the optimal size of the `–m` argument. The more the value of the `–m` argument, the more number of map tasks will run in parallel, which results in the increase of the load in the database server as well as the time of the import process. If the `–m` argument has a lower value, it would mean that the resource of your cluster is not fully utilized.

By default, Sqoop will use the primary key column of the input table to split work between the map tasks.

Let us consider an example: If we have a table that has a primary key column (`id`), where the minimum value of the `id` column is `0` and the maximum value is `10000`, and the value of the `–m` argument is `10`, Sqoop will internally execute 10 map tasks, and each task will execute the SQL statement of the following form:

```
select * from tableName where id>=min AND id < max
```

Following are the possible ranges (`min, max`) for the current example:

- `(0, 1000)`
- `(1000, 2000)`
- `(2000, 3000)`
- `(3000, 4000)`
- `(5000, 6000)`
- `(7000, 8000)`
- `(8000, 9000)`
- `(9000, 10000)`

If the table doesn't contain the primary key column, we need to explicitly specify the splitting column in the `–split-by` argument.

It is always recommended to use only those columns in `–split-by` whose values are equally distributed across its range. If the values of the columns are not equally distributed, some map tasks will finish much faster than other tasks.

The following example controls the import parallelism by using 16 parallel tasks:

**Query 11**:

```
$ bin/sqoop import –connect jdbc:mysql://localhost:3306/db1 –username
root –password password –table tableName –target-dir /user/abc/tableName
-m 16
```

## Incremental import (Simple)

Incremental import means importing the new version of records or the latest inserted records from the RDBMS table into HDFS.

### Getting ready

We can control the incremental import by using the arguments listed in the following table:

| Parameter/argument | Description |
|---|---|
| `--check-column <column-name>` | The value of this column is used to determine the rows to be imported during the import process. |
| `--incremental <incremental-type>` | Specifies the type of incremental mode. Possible values are `append` and `lastmodified`. |

| Parameter/argument | Description |
|---|---|
| `--last-value <value>` | Specifies the last value or the maximum value of the `check` column from the previous import. All the records whose `check` column value is greater than the value of the `–last-value` argument will be imported to HDFS. |

## How to do it...

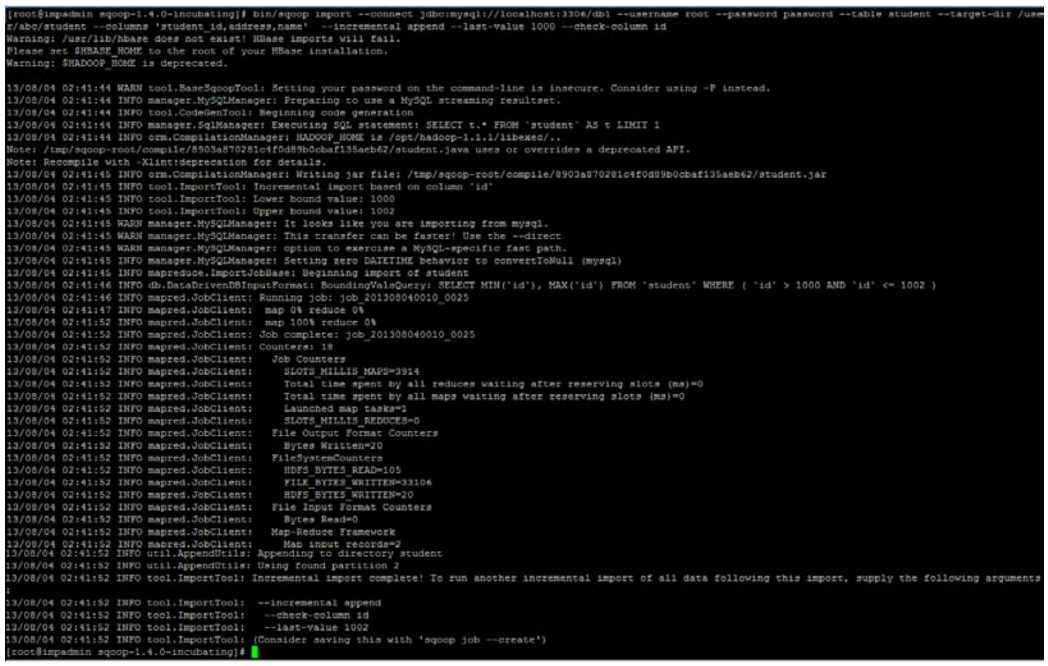
Let's see how to perform an incremental import:

1. Our first step is to append new records into imported records.

   **Query 12**:

   ```
   $bin/sqoop import –connect jdbc:mysql://localhost:3306/db1 –
   username root –password password –table student –target-dir /user/
   abc/student –columns "student_id,address,name"  --incremental
   append –last-value 1000 –check-column id
   ```

   We run the query as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop import --connect jdbc:mysql://localhost:3306/db1 --username root --password password --table student --target-dir /use
r/abc/student --columns 'student_id,address,name'  --incremental append --last-value 1000 --check-column id
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: $HADOOP_HOME is deprecated.

13/08/04 02:41:44 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 02:41:44 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 02:41:44 INFO tool.CodeGenTool: Beginning code generation
13/08/04 02:41:44 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `student` AS t LIMIT 1
13/08/04 02:41:44 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/8903a870281c4f0d89b0cbaf135aeb62/student.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 02:41:45 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/8903a870281c4f0d89b0cbaf135aeb62/student.jar
13/08/04 02:41:45 INFO tool.ImportTool: Incremental import based on column `id`
13/08/04 02:41:45 INFO tool.ImportTool: Lower bound value: 1000
13/08/04 02:41:45 INFO tool.ImportTool: Upper bound value: 1002
13/08/04 02:41:45 WARN manager.MySQLManager: It looks like you are importing from mysql.
13/08/04 02:41:45 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
13/08/04 02:41:45 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
13/08/04 02:41:45 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
13/08/04 02:41:45 INFO mapreduce.ImportJobBase: Beginning import of student
13/08/04 02:41:46 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`id`), MAX(`id`) FROM `student` WHERE ( `id` > 1000 AND `id` <= 1002 )
13/08/04 02:41:46 INFO mapred.JobClient: Running job: job_201308040010_0025
13/08/04 02:41:47 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 02:41:52 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 02:41:52 INFO mapred.JobClient: Job complete: job_201308040010_0025
13/08/04 02:41:52 INFO mapred.JobClient: Counters: 18
13/08/04 02:41:52 INFO mapred.JobClient:   Job Counters
13/08/04 02:41:52 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=3914
13/08/04 02:41:52 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 02:41:52 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 02:41:52 INFO mapred.JobClient:     Launched map tasks=1
13/08/04 02:41:52 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 02:41:52 INFO mapred.JobClient:   File Output Format Counters
13/08/04 02:41:52 INFO mapred.JobClient:     Bytes Written=20
13/08/04 02:41:52 INFO mapred.JobClient:   FileSystemCounters
13/08/04 02:41:52 INFO mapred.JobClient:     HDFS_BYTES_READ=105
13/08/04 02:41:52 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=33106
13/08/04 02:41:52 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=20
13/08/04 02:41:52 INFO mapred.JobClient:   File Input Format Counters
13/08/04 02:41:52 INFO mapred.JobClient:     Bytes Read=0
13/08/04 02:41:52 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 02:41:52 INFO mapred.JobClient:     Map input records=2
13/08/04 02:41:52 INFO util.AppendUtils: Appending to directory student
13/08/04 02:41:52 INFO util.AppendUtils: Using found partition 2
13/08/04 02:41:52 INFO tool.ImportTool: Incremental import complete! To run another incremental import of all data following this import, supply the following arguments
:
13/08/04 02:41:52 INFO tool.ImportTool:  --incremental append
13/08/04 02:41:52 INFO tool.ImportTool:   --check-column id
13/08/04 02:41:52 INFO tool.ImportTool:   --last-value 1002
13/08/04 02:41:52 INFO tool.ImportTool: (Consider saving this with 'sqoop job --create')
[root@impadmin sqoop-1.4.0-incubating]# 
```

2. Our next step is to append both new records and updated records into already imported records.

   **Query 13**:

   ```
   bin/sqoop import –connect jdbc:mysql://localhost:3306/db1 –
   username root –password password –table student –target-dir /user/
   abc/student –columns "student_id,address,name"  --incremental
   lastmodified –last-value "2012-11-06 19:01:35"–check-column col4
   ```

3. Finally, we will put some light on how we can save the job information in the Sqoop metastore.

   **Query 14**:

   ```
   $ bin/sqoop job –create myjob-import –connect jdbc:mysql://
   localhost:3306/db1 –username root –password password–table student
   –target-dir /user/abc/student –columns "student_id,address,name"
   --incremental lastmodified –last-value "2012-11-06 19:01:35"–
   check-column col4
   ```

## How it works...

In the append mode, we use the incremental column (generally ID) as a value of the `–check-column` argument. Append mode will be useful in those scenarios where continuous rows are added into the table with increasing rows ID.

All the records that have a higher value of the row ID (incremental column or ID) than the value of the `–last-value` argument will be appended in the existing data.

For example, if the value of the `–last-value` argument is 100 and the row ID is used as the value of `–check-column`, all the rows that have the value of row ID greater than 100 will be appended to previously imported data.

In the append mode, only the newly added record will be imported to HDFS. All modified/updated records are not reimported to HDFS.

Query 12 performs the incremental import and appends all the new rows to the already imported 1,000 rows.

Query 12 will print the following lines on the console:

```
INFO tool.ImportTool:  --incremental append
INFO tool.ImportTool:  --check-column id
INFO tool.ImportTool:  --last-value 1002
```

The output shows that the ID of the last imported record is `1002`, which means a total of 1002 records of the `student` table are imported to HDFS, where `id` is the incremental column.

By default, Sqoop will overwrite the content of the existing directory with a new set of records. But if we use an `–append` argument, Sqoop first copies the data into the temporary directory and then renames the files into the given target directory to avoid conflict with the existing files in the target directory.

In the incremental mode, only those rows are imported to HDFS that have a higher value of the timestamp column than the value specified in `–last-value` arguments.

In incremental import, both updated records and newly added records will be imported to HDFS.

Query 13 shows an example in which the input table consists of four columns, namely, `student_id`, `address`, `name`, and `col4`. Here, `col4` contains the inserted/updated time of the record, and we are using `col4` as `–check-column`.

Query 13 will print the following lines on the console:

```
INFO tool.ImportTool:  --incremental append
INFO tool.ImportTool:   --check-column col4
INFO tool.ImportTool:   --last-value 2013-04-18 00:19:49.0
```

The preceding output gives the value of `–last-value` arguments, which should be used as the value of `–last-value` for a subsequent import.

The advantage of using the `lastmodified` mode is that it will import both the newly added and updated records, but the input table has to maintain an extra column (timestamp) for this feature.

Sqoop provides a mechanism to save the job. A saved job stores the configuration information required to run the same job at a later time.

In case of an incremental import, a user is required to run the same query multiple times and also needs to record the value of the `–last-value` argument for the next run. By using the saved job mechanism, we can save the job information at the Sqoop metastore. The metastore also includes the last value of the incremental column or the timestamp column.

We can save the job information by using the arguments shown in the following table:

| Parameter | Description |
|---|---|
| `--create <job-id>` | Creates a new saved job. |
| `--delete <job-id>` | Deletes a saved job. |
| `--exec <job-id>` | Executes the saved job. |
| `--show <job-id>` | Shows the parameters of the saved job. |
| `--list` | Lists all the saved jobs. |

Query 14 covered an example to create a new saved job.

The following command is used to view the list of available jobs:

```
$ bin/sqoop job -list
Available jobs:
  myjob
```

The following command is used to execute the saved job:

```
$bin/sqoop job -exec myjob
INFO tool.CodeGenTool: Beginning code generation
```

The following command is used to show the parameters of the saved job:

```
$ bin/sqoop job -show myjob
Job: myjob
Tool: import
incremental.last.value = 2011-11-24 15:09:38.0
```

## Populating the HBase table (Simple)

Before explaining the Apache HBase basic commands, let's have an overview of the HBase and its table structure.

HBase is a NoSQL, multidimensional, sparse, and a horizontally scalable database modeled after Google BigTable. HBase is built on the top of Hadoop, which means that it relies on Hadoop and integrates very well with the MapReduce framework. Hadoop provides the following benefits to HBase:

- A distributed data store running on top of the commodity hardware
- Data redundancy
- Fault tolerance

### Getting ready

The prerequisites are as follows:

1. Running the HBase cluster.
2. The environment variable `$HBASE_HOME` is set.

## How to do it...

Let's see the basic commands of HBase:

1. We will go through the `create`, `get`, `put`, `disable`, and `delete` commands of HBase.

## How it works...

The following command starts the HBase shell:

```
$ bin/hbase shell
```

The table is created using the following command:

```
hbase(main):001:0> create 'tableName1','columnfamily1'
```

[ Table name and column families are required during table creation, while columns are created in runtime. ]

The following command is used to describe the table:

```
hbase(main):002:0> describe 'tableName1'
```

The following command is used to insert a row:

```
hbase(main):003:0> put 'tableName1','1','columnfamily1:col1','abc'
```

[ Here, `1` is the value of the row key and `col1` is a column created inside the column family, `columnfamily1`. ]

The following command scans the HBase table:

```
hbase(main):004:0> scan 'tableName1'
```

The following command gets the row by passing the row key:

```
hbase(main):005:0> get 'tableName1','1'
```

The following command drops the table:

```
hbase(main):006:0> disable 'tableName1'
hbase(main):007:0> drop 'tableName1'
```

## There's more...

HBase is also good for those use cases where users want to perform random read/write on TBs of data stored in one table.

In HBase, each table consists of rows and columns and all the columns in the HBase table belong to a particular column family.

Consider the following table structure, for example:

```
Student (Table Name)
1 (Row)
->name (Column Family)
    ->firstName (column1)
      -> Value
      ->TimeStamp
    ->LastName (column1)
''''''''''' -> Value
      ->TimeStamp
2 (Row)
->name (Column Family)
    ->firstName (column1)
      -> Value
      ->TimeStamp
    ->LastName (column1)
-> Value
      ->TimeStamp
    ->MiddleName (column1)
      -> Value
      ->TimeStamp
```

This example explained the two important features of HBase:

- Its multidimensional structure
- As depicted in the previous example, a given table can have any number of columns in each column family or null at all

## Importing data into HBase (Intermediate)

So far, all the Sqoop statements that we have looked at were dumped from the RDBMS table into HDFS. Now we are focusing on importing the RDBMS data to HBase.

### Getting ready

The following table shows a list of HBase arguments that we have to use to import the RDBMS data to HBase:

| Parameter / Argument | Description |
|---|---|
| `--coulmn-family <columnfamily-name>` | Specifies a column family of the target HBase table. |
| `--hbase-create-table` | By using `–hbase-create-table`, Sqoop will create the missing table and the column family in HBase before executing the MapReduce job. |
| `--hbase-row-key <column>` | Specifies which input column to use as an HBase row key. |
| `--hbase-table <table-name>` | Specifies the name of the target HBase table. |

### How to do it...

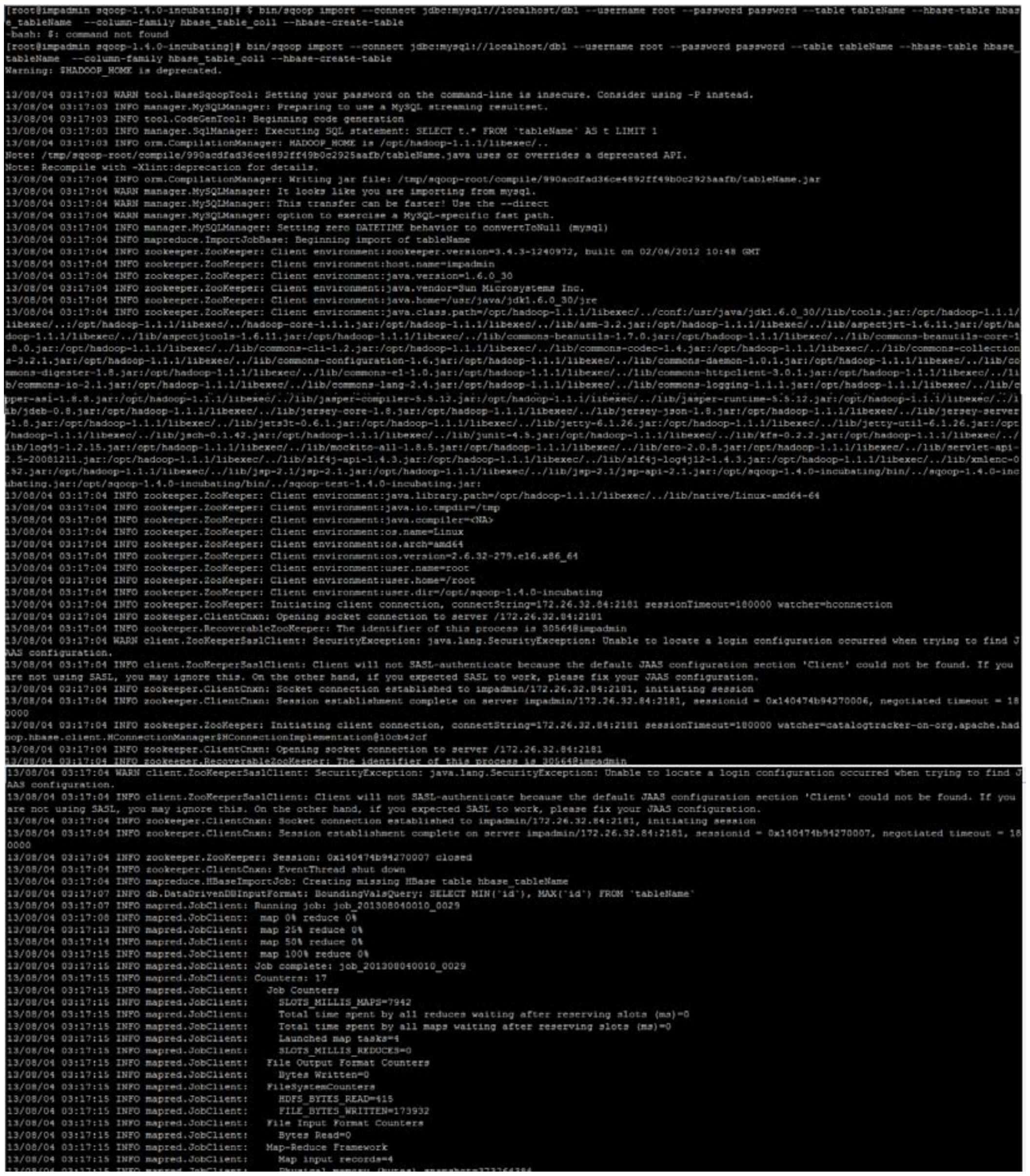
Let's see how to import data into HBase:

1. First we will explain the examples of importing the primary key table (a table that has the primary key) into HBase.

   **Query 15**:

   ```
   $ bin/sqoop import –connect jdbc:mysql://localhost/db1 –username
   root –password password –table tableName –hbase-table hbase_
   tableName  --column-family hbase_table_col1 –hbase-creat
   ```

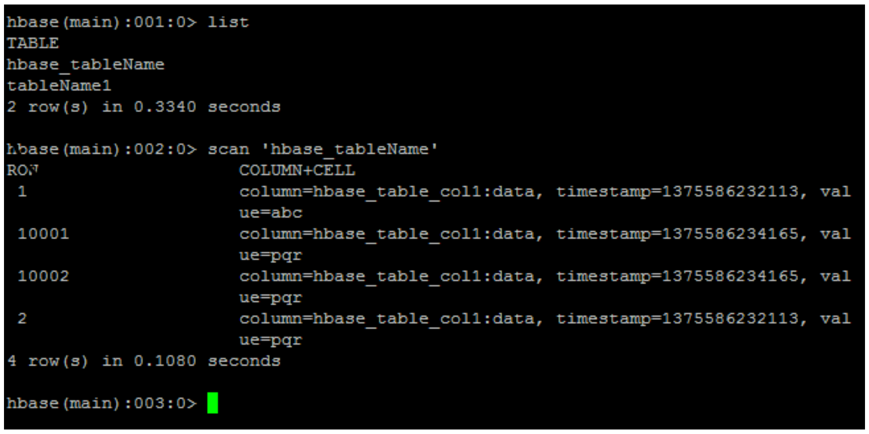
Run the query as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# $ bin/sqoop import --connect jdbc:mysql://localhost/db1 --username root --password password --table tableName --hbase-table hbas
e_tableName  --column-family hbase_table_col1 --hbase-create-table
-bash: $: command not found
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop import --connect jdbc:mysql://localhost/db1 --username root --password password --table tableName --hbase-table hbase_
tableName  --column-family hbase_table_col1 --hbase-create-table
Warning: $HADOOP_HOME is deprecated.

13/08/04 03:17:03 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 03:17:03 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 03:17:03 INFO tool.CodeGenTool: Beginning code generation
13/08/04 03:17:03 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 03:17:03 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/990acdfad36ce4892ff49b0c2925aafb/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 03:17:04 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/990acdfad36ce4892ff49b0c2925aafb/tableName.jar
13/08/04 03:17:04 WARN manager.MySQLManager: It looks like you are importing from mysql.
13/08/04 03:17:04 WARN manager.MySQLManager: This transfer can be faster! Use the --direct
13/08/04 03:17:04 WARN manager.MySQLManager: option to exercise a MySQL-specific fast path.
13/08/04 03:17:04 INFO manager.MySQLManager: Setting zero DATETIME behavior to convertToNull (mysql)
13/08/04 03:17:04 INFO mapreduce.ImportJobBase: Beginning import of tableName
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.3-1240972, built on 02/06/2012 10:48 GMT
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:host.name=impadmin
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.version=1.6.0_30
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.vendor=Sun Microsystems Inc.
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.6.0_30/jre
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.class.path=/opt/hadoop-1.1.1/libexec/../conf:/usr/java/jdk1.6.0_30//lib/tools.jar:/opt/hadoop-1.1.1/
libexec/..:/opt/hadoop-1.1.1/libexec/../hadoop-core-1.1.1.jar:/opt/hadoop-1.1.1/libexec/../lib/asm-3.2.jar:/opt/hadoop-1.1.1/libexec/../lib/aspectjrt-1.6.11.jar:/opt/ha
doop-1.1.1/libexec/../lib/aspectjtools-1.6.11.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-beanutils-1.7.0.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-beanutils-core-1
.8.0.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-cli-1.2.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-codec-1.4.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-collection
s-3.2.1.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-configuration-1.6.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-daemon-1.0.1.jar:/opt/hadoop-1.1.1/libexec/../lib/co
mmons-digester-1.8.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-el-1.0.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-httpclient-3.0.1.jar:/opt/hadoop-1.1.1/libexec/../li
b/commons-io-2.1.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-lang-2.4.jar:/opt/hadoop-1.1.1/libexec/../lib/commons-logging-1.1.1.jar:/opt/hadoop-1.1.1/libexec/../lib/c
pper-asl-1.8.8.jar:/opt/hadoop-1.1.1/libexec/../lib/jasper-compiler-5.5.12.jar:/opt/hadoop-1.1.1/libexec/../lib/jasper-runtime-5.5.12.jar:/opt/hadoop-1.1.1/libexec/../l
ib/jdeb-0.8.jar:/opt/hadoop-1.1.1/libexec/../lib/jersey-core-1.8.jar:/opt/hadoop-1.1.1/libexec/../lib/jersey-json-1.8.jar:/opt/hadoop-1.1.1/libexec/../lib/jersey-server
-1.8.jar:/opt/hadoop-1.1.1/libexec/../lib/jets3t-0.6.1.jar:/opt/hadoop-1.1.1/libexec/../lib/jetty-6.1.26.jar:/opt/hadoop-1.1.1/libexec/../lib/jetty-util-6.1.26.jar:/opt
/hadoop-1.1.1/libexec/../lib/jsch-0.1.42.jar:/opt/hadoop-1.1.1/libexec/../lib/junit-4.5.jar:/opt/hadoop-1.1.1/libexec/../lib/kfs-0.2.2.jar:/opt/hadoop-1.1.1/libexec/../
lib/log4j-1.2.15.jar:/opt/hadoop-1.1.1/libexec/../lib/mockito-all-1.8.5.jar:/opt/hadoop-1.1.1/libexec/../lib/oro-2.0.8.jar:/opt/hadoop-1.1.1/libexec/../lib/servlet-api-
2.5-20081211.jar:/opt/hadoop-1.1.1/libexec/../lib/slf4j-api-1.4.3.jar:/opt/hadoop-1.1.1/libexec/../lib/slf4j-log4j12-1.4.3.jar:/opt/hadoop-1.1.1/libexec/../lib/xmlenc-0
.52.jar:/opt/hadoop-1.1.1/libexec/../lib/jsp-2.1/jsp-2.1.jar:/opt/hadoop-1.1.1/libexec/../lib/jsp-2.1/jsp-api-2.1.jar:/opt/sqoop-1.4.0-incubating/bin/../sqoop-1.4.0-inc
ubating.jar:/opt/sqoop-1.4.0-incubating/bin/../sqoop-test-1.4.0-incubating.jar:
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.library.path=/opt/hadoop-1.1.1/libexec/../lib/native/Linux-amd64-64
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.io.tmpdir=/tmp
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:java.compiler=<NA>
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:os.name=Linux
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:os.arch=amd64
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:os.version=2.6.32-279.el6.x86_64
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:user.name=root
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:user.home=/root
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Client environment:user.dir=/opt/sqoop-1.4.0-incubating
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=172.26.32.84:2181 sessionTimeout=180000 watcher=hconnection
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: Opening socket connection to server /172.26.32.84:2181
13/08/04 03:17:04 INFO zookeeper.RecoverableZooKeeper: The identifier of this process is 30564@impadmin
13/08/04 03:17:04 WARN client.ZooKeeperSaslClient: SecurityException: java.lang.SecurityException: Unable to locate a login configuration occurred when trying to find J
AAS configuration.
13/08/04 03:17:04 INFO client.ZooKeeperSaslClient: Client will not SASL-authenticate because the default JAAS configuration section 'Client' could not be found. If you
are not using SASL, you may ignore this. On the other hand, if you expected SASL to work, please fix your JAAS configuration.
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: Socket connection established to impadmin/172.26.32.84:2181, initiating session
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: Session establishment complete on server impadmin/172.26.32.84:2181, sessionid = 0x140474b94270006, negotiated timeout = 18
0000
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=172.26.32.84:2181 sessionTimeout=180000 watcher=catalogtracker-on-org.apache.had
oop.hbase.client.HConnectionManager$HConnectionImplementation@10cb42cf
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: Opening socket connection to server /172.26.32.84:2181
13/08/04 03:17:04 INFO zookeeper.RecoverableZooKeeper: The identifier of this process is 30564@impadmin
13/08/04 03:17:04 WARN client.ZooKeeperSaslClient: SecurityException: java.lang.SecurityException: Unable to locate a login configuration occurred when trying to find J
AAS configuration.
13/08/04 03:17:04 INFO client.ZooKeeperSaslClient: Client will not SASL-authenticate because the default JAAS configuration section 'Client' could not be found. If you
are not using SASL, you may ignore this. On the other hand, if you expected SASL to work, please fix your JAAS configuration.
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: Socket connection established to impadmin/172.26.32.84:2181, initiating session
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: Session establishment complete on server impadmin/172.26.32.84:2181, sessionid = 0x140474b94270007, negotiated timeout = 18
0000
13/08/04 03:17:04 INFO zookeeper.ZooKeeper: Session: 0x140474b94270007 closed
13/08/04 03:17:04 INFO zookeeper.ClientCnxn: EventThread shut down
13/08/04 03:17:04 INFO mapreduce.HBaseImportJob: Creating missing HBase table hbase_tableName
13/08/04 03:17:07 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(`id`), MAX(`id`) FROM `tableName`
13/08/04 03:17:07 INFO mapred.JobClient: Running job: job_201308040010_0029
13/08/04 03:17:08 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 03:17:13 INFO mapred.JobClient:  map 25% reduce 0%
13/08/04 03:17:14 INFO mapred.JobClient:  map 50% reduce 0%
13/08/04 03:17:15 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 03:17:15 INFO mapred.JobClient: Job complete: job_201308040010_0029
13/08/04 03:17:15 INFO mapred.JobClient: Counters: 17
13/08/04 03:17:15 INFO mapred.JobClient:   Job Counters
13/08/04 03:17:15 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=7942
13/08/04 03:17:15 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 03:17:15 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 03:17:15 INFO mapred.JobClient:     Launched map tasks=4
13/08/04 03:17:15 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 03:17:15 INFO mapred.JobClient:   File Output Format Counters
13/08/04 03:17:15 INFO mapred.JobClient:     Bytes Written=0
13/08/04 03:17:15 INFO mapred.JobClient:   FileSystemCounters
13/08/04 03:17:15 INFO mapred.JobClient:     HDFS_BYTES_READ=415
13/08/04 03:17:15 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=173932
13/08/04 03:17:15 INFO mapred.JobClient:   File Input Format Counters
13/08/04 03:17:15 INFO mapred.JobClient:     Bytes Read=0
13/08/04 03:17:15 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 03:17:15 INFO mapred.JobClient:     Map input records=4
13/08/04 03:17:15 INFO mapred.JobClient:     Physical memory (bytes) snapshot=173264384
```

The output table in HBase would look like the following screenshot:



```
hbase(main):001:0> list
TABLE
hbase_tableName
tableName1
2 row(s) in 0.3340 seconds

hbase(main):002:0> scan 'hbase_tableName'
ROW                   COLUMN+CELL
 1                    column=hbase_table_col1:data, timestamp=1375586232113, val
                      ue=abc
 10001                column=hbase_table_col1:data, timestamp=1375586234165, val
                      ue=pqr
 10002                column=hbase_table_col1:data, timestamp=1375586234165, val
                      ue=pqr
 2                    column=hbase_table_col1:data, timestamp=1375586232113, val
                      ue=pqr
4 row(s) in 0.1080 seconds

hbase(main):003:0>
```

**Query 16**:

```
$ bin/sqoop import -connect jdbc:mysql://localhost/db1 -username
root -password password -table tableName -hbase-table hbase_
tableName -columns column1,column2 -column-family hbase_table_col1
-hbase-create-table
```

2. Next we will cover the examples to import a non-primary key table into HBase.

   **Query 17**:

```
$ bin/sqoop import -connect jdbc:mysql://localhost/db1 -username
root -password root -table tableName -hbase-table hbase_tableName
-columns column1,column2 -column-family hbase_table_col1-hbase-
row-key column1 -hbase-create-table
```

## How it works...

This section discusses sample examples to import the RDBMS data into HBase.

### Importing a primary key table into HBase

In Sqoop, the `--hbase-row-key` argument is used to specify which input column we want to use as the HBase row key. If `–hbase-row-key` and `–split-by` are not used, Sqoop itself tries to identify the primary key of the given RDBMS table and uses that column as the HBase row key.

Query 15 shows an example to import all the columns of the input table into HDFS.

Query 16 covers an example to import the selected columns of the given table into HBase.

## Importing a non-primary key table into HBase

We can import a non-primary key table into HBase by using the `–hbase-row-key` argument.

Query 17 shows an example to import a non-primary key table into HBase, where `column1` is a non-primary key column.

It is always recommended to use either the primary column as the HBase row-key or a unique column as the HBase row-key; otherwise, there may be risk of data loss.

For example, consider a non-primary key table `table1` shown as follows:

| col1 | col2 | col3 |
|---|---|---|
| 1 | ABS | PQR |
| 1 | ABS | PRW |
| 2 | AAA | PQR |

If we import `table1` into HBase by using `col1` as the `–split-by` column or the `–hbase-row-key` column, only two records are visible in the HBase table. The reason behind the data loss is that the HBase creates a primary index on the basis of the row-key, and if the same value of the row-key column appears again, HBase will create a new version of that row.

The output in HBase will look as follows:

```
Row key (col1)     Column Family (cf1) and column (col2 and col3)


1                  column=cf1:col2, timestamp=1366327462514, value=ABS

1                  column=cf1:col3, timestamp=1366327462514, value=PRW


2                  column=cf1:col2, timestamp=1366327462932, value=AAA

2                  column=cf1:col3, timestamp=1366327462932, value=PQR
```

The printed output contains only two rows even though all the three rows are imported in the import process. Internally, HBase has created two versions of same row keys and only the latest timestamp value of the row key is returned in the output. But HBase also provides a mechanism to view all the versions of the given row key.

## Populating the Hive table (Simple)

Before explaining the Hive basic commands, I would like to give an overview of Hive.

Hive is a data warehouse infrastructure built on top of Hadoop, which is heavily used for data summarization, analysis and ad hoc querying. It was created by Facebook and then contributed back to the Hadoop ecosystem as a Hadoop subproject. Hive is not designed to handle online transactions and does not generate real-time results because Hive queries submit the MapReduce job on Hadoop which then operates on files stored in HDFS.

### Getting ready

We assume you have Hive installed and the `$HIVE_HOME` environment set on your machine. This section will focus on some common Hive commands.

### How to do it...

Let's start with Hive's basic commands and their syntax:

1. We will go through `create`, `delete`, `list`, `load` and `describe` commands.

### How it works...

First, start the Hive Console via the following command:

```
>$ cd $HIVE_HOME;bin/hive
```

Then create a table in Hive:

```
hive> create table student (id INT, name STRING);
```

Get a list of tables:

```
hive> show tables;
```

And then describe the table:

```
hive> describe student;
```

Load data from the local filesystem into the Hive table:

```
hive> load data local inpath'./examples/files/kv1.txt' overwrite into
table student;
```

Load data from HDFS into the Hive table:

```
hive> load data inpath'/user/username/kv1.txt' overwrite into table
student;
```

By default, Hive will assume *Ctrl* + *A* as the field delimiter and newline as the line delimiter. If the input file doesn't contain the default field and the line delimiter, we need to specify the field delimiter and the line separator at the time of table creation.

Create a table by specifying the field delimiter and the row delimiter:

```
Hive> CREATE TABLE student(id int, name string)ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','STORED AS SEQUENCEFILE;
```

Select or print all the rows of the given table:

```
hive> select * from student;
```

Put in the `where` conditions:

```
hive> select id from student where name = "abc";
```

And then drop the table:

```
hive> drop table student;
```

## There's more...

Hive has the following important features:

- Hive supports indexing to provide acceleration.
- Support for different storage types such as plain text, RCFiles, HBase, and others.
- Hive stores metadata in an RDBMS, which reduces a significant amount of time to perform semantic checks during the query execution.
- Hive can operate on compressed data stored into the Hadoop ecosystem.
- Built-in **User Defined Functions** (**UDFs**) to manipulate dates, strings, and other data-mining tools. If none serves our need, we can create our own UDFs.
- Hive supports SQL-like queries (Hive QL) that are implicitly converted into MapReduce jobs.

## Importing data into Hive (Simple)

So far, we have worked on dumping the RDBMS data into HDFS or HBase. This recipe will focus on common arguments that are required to dump the data into Hive and also explains some examples to move the RDBMS data into Hive.

### Getting ready

The following table shows the list of arguments that we have to use to move the RDBMS data to Hive:

| **Parameters** | **Description** |
|---|---|
| `--hive-home <dir>` | This parameter is used to override the `$HIVE_HOME` environment variable. |
| `--hive-import` | This parameter informs Sqoop that a user wants to import data into Hive. |
| `--hive-overwrite` | This is used if a user wants to overwrite the content of the existing table. |
| `--create-hive-table` | If this argument is used, Sqoop will throw an error if an input-output table exists in Hive. |
| `--hive-table <table-name>` | Sets the name of the output table (the Hive table). |
| `--hive-drop-import-delims` | Removes `'\n'`,`'\01'`, and `'\r'` from the value of the input fields. The default field delimiter is `^A` and the default line delimiter is `\n`. |
| `--hive-delims-replacement` | Replaces `'\n'`,`'\01'`, and `'\r'` from the value of the input fields with user-defined delimiters. |

### How to do it...

Let's see how to import data into Hive:

1. We will cover approaches to import the RDBMS data into Hive.

## How it works...

There are two ways to import the data into Hive. The first way is to import the RDBMS table into HDFS and then create an external table in Hive. This external table will use the HDFS location as a data reference point.

Consider the following example:

We are importing a `student` table into HDFS at `/user/username/student`. Once the data is imported in HDFS, we can create an external table in Hive by using the following command:

```
Hive> CREATE EXTERNAL TABLE student(id int, name string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n    STORED AS TEXTFILE
LOCATION '/user/username/student';
```

By using the preceding command, users can access the HDFS data through Hive.

The second way is by using the `–hive-table` and `–create-hive-table` arguments.

If we use both `–hive-table` and `–create-hive-table` arguments in the Sqoop statement, Sqoop itself will create the table in Hive and load the data into the Hive warehouse directory. The only difference between HDFS and a Hive import is that the Hive import performs post processing that will load the HDFS data into the Hive warehouse directory.

As we know, if the value of input columns contains the default field and the line delimiter, we can use `enclosed-by` and `escaped-by` characters in the Sqoop statement to avoid ambiguity while parsing the data. But, hive does not support escaping of the newline character, and also it doesn't support enclosing of the input field value. However, we can remove the ambiguous parsing by following this approach:

1. Use the `–hive-drop-import-delims` option to remove the field delimiter and line terminator characters from the field value. We can also use `–hive-delims-replacements` to replace the field delimiter and the line terminator with a user-defined string.
2. If the output table already exists in Hive, new data will be appended to the existing table. But we can change this default behavior by using the `–hive-overwrite` argument. If the user uses the `–hive-overwrite` argument, the existing data will overwrite with new data.

## There's more...

This section contains sample queries to transfer the RDBMS data into Hive.

### Importing a primary key table into Hive

During the import process, Sqoop will use the primary key column to divide the MapReduce job into multiple tasks.

The Sqoop statement to load RDBMS data into Hive is as follows:

```
$ bin/sqoop import  --connect jdbc:mysql://localhost:3306/db1 –username
root –password password–table tableName  --hive-table tableName –create-
hive-table –hive-import –hive-home path/to/hive_home
```

### Importing a non-primary key table into Hive

If the input table doesn't contain a primary key column, the user has to manually specify the `split-by` column in the Sqoop statement. Sqoop will use the value of the `split-by` argument to divide the job into multiple tasks.

The Sqoop `import` command, using the `–split-by` argument, is as follows:

```
$ bin/sqoop import  --connect jdbc:mysql://localhost:3306/db1 –username
root –password password–table tableName  --hive-table tableName –create-
hive-table –hive-import –hive-home path/to/hive_home –split-by column_
name
```

## The exporting process (Intermediate)

As mentioned in the *Working with the import process* recipe, Sqoop has two important features: importing and exporting data. So far, we have only worked on the import feature. This recipe will give us an overview of the Sqoop export process.

Exporting a process means moving the HDFS data back to the RDBMS table. The output table must exist in RDBMS before running the export job. Sqoop reads the records from the delimited text file one-by-one, translates it into the RDBMS `INSERT` command, and then inserts it into the output table. If the output table contains a primary key, the value of the primary key column in the input file must be unique and not null; otherwise, the export process will stop due to the violation of primary key constraints.

Like the import process, the export process also executes the MapReduce job to achieve parallelism.

## Getting ready

The following table shows the list of arguments required to transfer the process data from Hadoop to RDBMS:

| Parameters | Description |
|---|---|
| `--direct` | Use the direct mode to perform the export quickly. Note that it is only supported for MySQL. |
| `--export-dir<dir>` | The location of input files in HDFS. |
| `--table <table-name>` | Name of the output table (the RDBMS table). |
| `-m,--num-mappers <n>` | Refers to the number of map tasks. |
| `--update-mode <mode>` | Specifies how updates are performed when new rows are found with non-matching keys in the database. Legal values for the mode include `updateonly` (default) and `allowinsert`. |
| `--update-key <col-name>` | The value of this column is used to identify the records that a user wants to update during the update mode. Use a comma-separated list of columns if there is more than one column. |
| `--staging-table <staging-table-name>` | Specifies the name of the staging table. The staging table is used to stage the data before inserting it into the destination table. |
| `--clear-staging-table` | This argument is used to clean the data from the staging table. |

The `–table` and `–export-dir` arguments are mandatory arguments for the export process.

## How to do it...

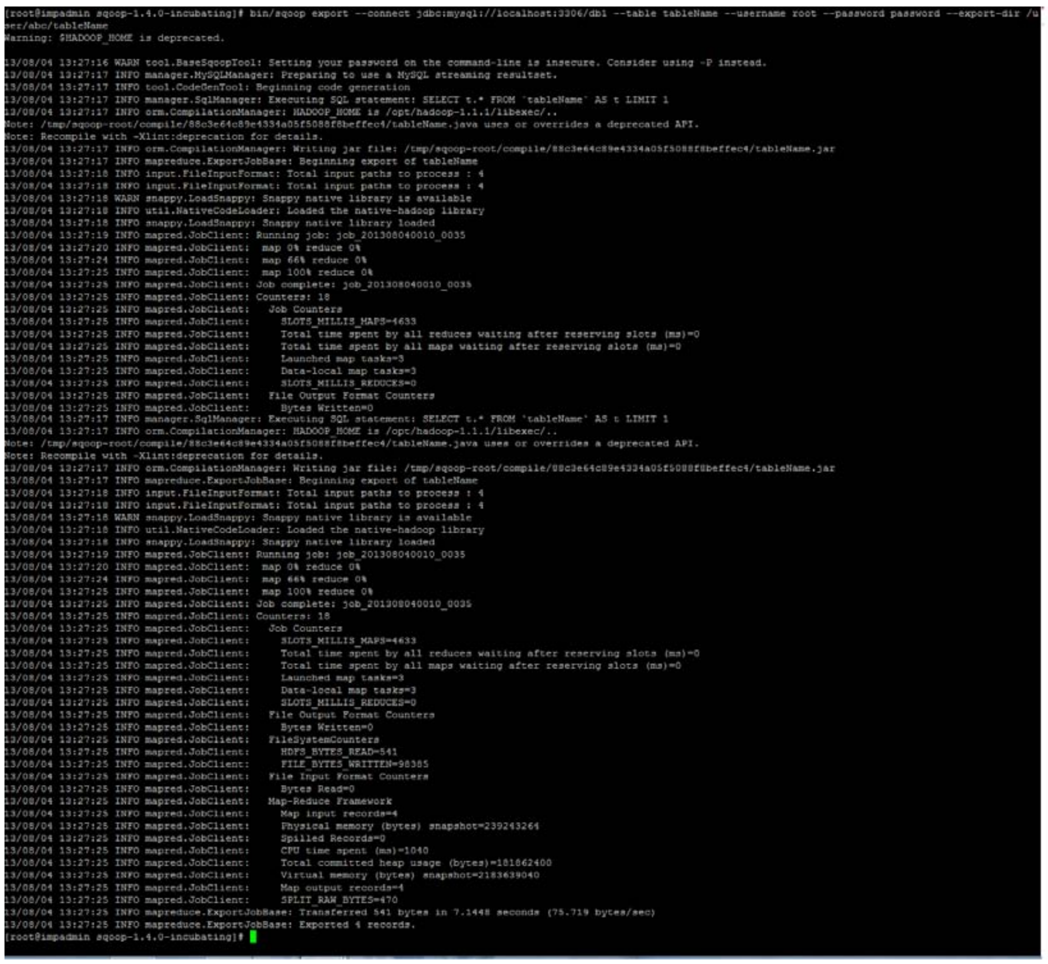
Let's see how to export data:

1. First, we will cover the architecture of the export process.
2. Second, we will cover an approach to avoid partial data export in case of a failover.
3. Third, we will explain the examples used to export the HDFS data back to RDBMS.

   **Query 18**:

   ```
   $ bin/sqoop export –connect jdbc:mysql://localhost:3306/db1 –table
   tableName  --username root --password password --export-dir /user/
   abc/tableName
   ```

Run the query as shown in the following screenshot:



```
[root@impadmin sqoop-1.4.0-incubating]# bin/sqoop export --connect jdbc:mysql://localhost:3306/db1 --table tableName --username root --password password --export-dir /u
ser/abc/tableName
Warning: $HADOOP_HOME is deprecated.

13/08/04 13:27:16 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
13/08/04 13:27:17 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
13/08/04 13:27:17 INFO tool.CodeGenTool: Beginning code generation
13/08/04 13:27:17 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 13:27:17 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/88c3e64c89e4334a05f5088f8beffec4/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 13:27:17 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/88c3e64c89e4334a05f5088f8beffec4/tableName.jar
13/08/04 13:27:17 INFO mapreduce.ExportJobBase: Beginning export of tableName
13/08/04 13:27:18 INFO input.FileInputFormat: Total input paths to process : 4
13/08/04 13:27:18 INFO input.FileInputFormat: Total input paths to process : 4
13/08/04 13:27:18 WARN snappy.LoadSnappy: Snappy native library is available
13/08/04 13:27:18 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/08/04 13:27:18 INFO snappy.LoadSnappy: Snappy native library loaded
13/08/04 13:27:19 INFO mapred.JobClient: Running job: job_201308040010_0035
13/08/04 13:27:20 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 13:27:24 INFO mapred.JobClient:  map 66% reduce 0%
13/08/04 13:27:25 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 13:27:25 INFO mapred.JobClient: Job complete: job_201308040010_0035
13/08/04 13:27:25 INFO mapred.JobClient: Counters: 18
13/08/04 13:27:25 INFO mapred.JobClient:   Job Counters
13/08/04 13:27:25 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=4633
13/08/04 13:27:25 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 13:27:25 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 13:27:25 INFO mapred.JobClient:     Launched map tasks=3
13/08/04 13:27:25 INFO mapred.JobClient:     Data-local map tasks=3
13/08/04 13:27:25 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 13:27:25 INFO mapred.JobClient:   File Output Format Counters
13/08/04 13:27:25 INFO mapred.JobClient:     Bytes Written=0
13/08/04 13:27:17 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `tableName` AS t LIMIT 1
13/08/04 13:27:17 INFO orm.CompilationManager: HADOOP_HOME is /opt/hadoop-1.1.1/libexec/..
Note: /tmp/sqoop-root/compile/88c3e64c89e4334a05f5088f8beffec4/tableName.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
13/08/04 13:27:17 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/88c3e64c89e4334a05f5088f8beffec4/tableName.jar
13/08/04 13:27:17 INFO mapreduce.ExportJobBase: Beginning export of tableName
13/08/04 13:27:18 INFO input.FileInputFormat: Total input paths to process : 4
13/08/04 13:27:18 INFO input.FileInputFormat: Total input paths to process : 4
13/08/04 13:27:18 WARN snappy.LoadSnappy: Snappy native library is available
13/08/04 13:27:18 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/08/04 13:27:18 INFO snappy.LoadSnappy: Snappy native library loaded
13/08/04 13:27:19 INFO mapred.JobClient: Running job: job_201308040010_0035
13/08/04 13:27:20 INFO mapred.JobClient:  map 0% reduce 0%
13/08/04 13:27:24 INFO mapred.JobClient:  map 66% reduce 0%
13/08/04 13:27:25 INFO mapred.JobClient:  map 100% reduce 0%
13/08/04 13:27:25 INFO mapred.JobClient: Job complete: job_201308040010_0035
13/08/04 13:27:25 INFO mapred.JobClient: Counters: 18
13/08/04 13:27:25 INFO mapred.JobClient:   Job Counters
13/08/04 13:27:25 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=4633
13/08/04 13:27:25 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/08/04 13:27:25 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/08/04 13:27:25 INFO mapred.JobClient:     Launched map tasks=3
13/08/04 13:27:25 INFO mapred.JobClient:     Data-local map tasks=3
13/08/04 13:27:25 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=0
13/08/04 13:27:25 INFO mapred.JobClient:   File Output Format Counters
13/08/04 13:27:25 INFO mapred.JobClient:     Bytes Written=0
13/08/04 13:27:25 INFO mapred.JobClient:   FileSystemCounters
13/08/04 13:27:25 INFO mapred.JobClient:     HDFS_BYTES_READ=541
13/08/04 13:27:25 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=98385
13/08/04 13:27:25 INFO mapred.JobClient:   File Input Format Counters
13/08/04 13:27:25 INFO mapred.JobClient:     Bytes Read=0
13/08/04 13:27:25 INFO mapred.JobClient:   Map-Reduce Framework
13/08/04 13:27:25 INFO mapred.JobClient:     Map input records=4
13/08/04 13:27:25 INFO mapred.JobClient:     Physical memory (bytes) snapshot=239243264
13/08/04 13:27:25 INFO mapred.JobClient:     Spilled Records=0
13/08/04 13:27:25 INFO mapred.JobClient:     CPU time spent (ms)=1040
13/08/04 13:27:25 INFO mapred.JobClient:     Total committed heap usage (bytes)=181862400
13/08/04 13:27:25 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=2183639040
13/08/04 13:27:25 INFO mapred.JobClient:     Map output records=4
13/08/04 13:27:25 INFO mapred.JobClient:     SPLIT_RAW_BYTES=470
13/08/04 13:27:25 INFO mapreduce.ExportJobBase: Transferred 541 bytes in 7.1448 seconds (75.719 bytes/sec)
13/08/04 13:27:25 INFO mapreduce.ExportJobBase: Exported 4 records.
[root@impadmin sqoop-1.4.0-incubating]#
```

4. Next we will go to the input parsing arguments.

   **Query 19**:

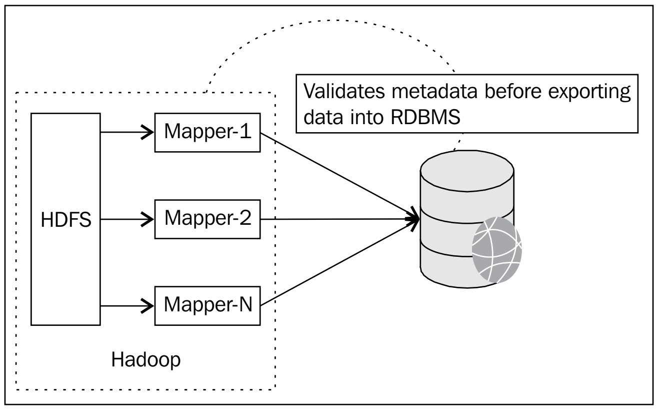
   ```
   $ bin/sqoop export –connect jdbc:mysql://localhost:3306/db1 –table
   tableName --username root --password password --export-dir /user/
   username/hdfsDir –input-fields-terminated-by '\t'  --input-lines-
   terminated-by '\n'
   ```

5. Finally, we will cover the overview of the export update command.

## How it works...

The export process depicted in the following diagram is broken down into two activities:

- Validating the metadata of the output RDBMS table
- Executing the MapReduce job to perform an actual data transfer task



Users can control the number of mappers independently from the number of files. It is always recommended using the optimal size of mappers to achieve better export performance.

By default, Sqoop will execute four tasks in parallel. The four parallel tasks are not an optimal case; the user has to decide the optimal size according to their setup. If the number of parallel tasks is less, your system may underutilize. On the other hand, if we use more numbers of mappers, it will increase the load on the RDBMS server.

Users can specify the number of map tasks in the Sqoop statement by using `–m` and `–num-mappers`.

As we know, Sqoop will split the export process into multiple tasks and all tasks will execute in parallel. If any job fails during the export operation, then only a subset of records will export to RDBMS because many tasks are running in parallel. Also, if we re-run the same export command, either the job would fail due to the already existing rows or the data would be duplicated into a table.

We can remove the previous problem by using the `–staging-table` argument; if we use the `–staging-table` argument, the data will stage into a staging table and then the staged data is moved to the destination table in a single transaction.

If we are using `–staging-table`, the staging table should be empty before running the export job or specify the `–clean-staging-table` argument in the Sqoop statement. If we specify the `–clean-staging-table` argument, Sqoop first cleans the staging table before running the export job.

The staging table must be identical to the destination table.

## Exporting the selected HDFS directory

Query 18 shows an example to export the HDFS directory into the RDBMS table.

The command from query 18 reads the files from HDFS at `/user/abc/tableName`, and then inserts the rows into the output table `tableName` of the `db1` database. An output table must exist before starting the actual export job, otherwise the export job fails. If any input row violates the primary key constraint or any other constraint, the export job will also fail.

## Inputting the parsing arguments

The following table lists the parsing arguments:

| **Arguments** | **Description** |
|---|---|
| `--input-enclosed-by <char>` | Specifies the required field enclosed. |
| `--input-escaped-by <char>` | Specifies the input escape character. |
| `--input-fields-terminated-by <char>` | Specifies the input field separator; this separator is used to split the input records into a field and then convert it into the `INSERT` statement. |
| `--input-lines-terminated-by <char>` | Specifies the input line terminator character or the record terminator character. |
| `--input-optionally-enclosed-by <char>` | Sets the field-enclosing character. |

Query 19 will export the input files into the output table, `tableName`, where `--input-fields-terminated` is used to specify the field terminator character, and `–input-lines-terminated` is used to specify the line terminated character.

> If the HDFS file doesn't contain default field delimiters `(,)` and a line terminator `(\n)`, we have to manually specify the field delimiters and line terminator characters.

## Exporting update commands

By default, Sqoop reads the records from the delimited text file one-by-one, translates it into the RDBMS `INSERT` command, and appends the rows into a table.

If we specify the `–update-key` arguments, Sqoop will modify or update the record into the output table. During the update mode, each input record will be converted into the RDBMS `UPDATE` statement and then update the existing dataset.

Let us consider an example; we have the RDBMS table whose definition is shown as follows:

```
CREATE TABLE invoice(
id INT NOT NULL PRIMARY KEY,
from VARCHAR(32),
to VARCHAR(32));
```

The input records in HDFS will look as follows:

```
1,ABC,PQR
2,CDF,ASD
```

If users run the export statement of the form `$ bin/sqoop export –connect jdbc:mysql://localhost:3306/db1 --username root --password password –table invoice–export-dir /user/username/hdfsDir –update-key id`, the input records will be translated into the following form:

```
UPDATE invoice SET from='ASD', to='QWE' WHERE id=1;
UPDATE invoice SET from='QWS', to='ACD' WHERE id=2;
```

> The Sqoop import command will execute successfully, even if no record is updated during the export process.

We can also specify the list of columns in `–update-key`. In that case, only those records are updated that satisfy all the required conditions.

Depending on the type of database, we can use the `–update-mode` argument; if the value of the `–update-mode` argument is `allowinsert`, the export command performs the following operations:

- Updates the record into the table if the record already exists
- Inserts the record into the table if the record doesn't exist

## Exporting data from Hive (Simple)

So far, we have focused on exporting data from HDFS; this section gives an overview of exporting data from Hive to RDBMS.

As we all know, the data of Hive resides in HDFS at `/user/hive/warehouse/tableName`. Also, Hive uses `'^A'` as the field delimiter and `'\n'` as the line terminator.

As such Sqoop doesn't provide any mechanism to export data directly from Hive to RDBMS, but there are some alternate ways to export the Hive table into RDBMS.

The dataset of the Hive table in HDFS will look like:

- `ABC ^A PQR`
- `CDF ^A ASD`

Here, `^A` is the field delimiter.

### How to do it...

Let's see how to export data from Hive:

1. It contains sample examples to transfer process data from Hive to RDBMS.

**Query 20**:

```
$ bin/sqoop export –connect jdbc:mysql://localhost/test_db –table invoice
--export-dir /user/hive/warehouse/invoice –username root –password
password –m 1 –input-fields-terminated-by '\001'
```

### How it works...

The command to create the invoice table in MySQL is as follows:

```
CREATE TABLE invoice(
id INT NOT NULL PRIMARY KEY,
from VARCHAR(32),
to VARCHAR(32));
```

Query 20 shows the command to export the Hive data into RDBMS, where `'\001'` is the octal representation of `^A`.

This example will read records from HDFS at `/user/hive/warehouse/invoice`, then split the record fields using `'\001'` field delimiter, and then generate the `INSERT` statement to insert the records into the RDBMS table.

## Using Sqoop connectors (Advanced)

As mentioned in the *Working with import process (Intermediate)* recipe, Sqoop supports the following types of databases:

- MySQL (direct mode support as well)
- Oracle
- SQL Server
- PostGre (direct mode support as well)
- DB2
- HSQLDB

The biggest advantage of using Sqoop is that we can write our own custom connector to support a different database as well. This recipe explains some third-party Sqoop connectors that are available on the market.

### How to do it...

Let's see how to add sqoop connectors.

1. We will cover five most popular third-party Sqoop connectors.

### How it works...

The Sqoop connectors are discussed as follows.

#### Couchbase Sqoop connector

The Couchbase Sqoop connector plugin allows us to connect to the Couchbase or Membase server and import data from Couchbase to the Hadoop ecosystem for processing.

We can download the Couchbase Hadoop connector from the following location:

`http://www.couchbase.com/develop/connectors/hadoop`

I am assuming that you have downloaded the Couchbase plugin and have extracted it at some location on your machine. We can automatically install the Couchbase plugin by just executing the following command in the `COUCHBASE_PLUGIN_HOME` directory:

```
$./install.sh SQOOP_HOME
```

Here, the `install.sh` file takes `$SQOOP_HOME` as the input.

Sqoop is primarily developed for importing data from RDBMS into Hadoop. The mandatory requirement for Sqoop is the name of the input table. The Couchbase plugin also takes the `–table` argument. In Couchbase, the `–table` argument is used to specify the type of tab stream a user wants to import into HDFS from Couchbase.

There are two possible values of the `–table` argument:

- `DUMP`: This causes all the keys currently in Couchbase to be read into HDFS
- `BACKFILL_<time_in_minutes>`: This streams all the keys' mutations for a given amount of time (in minutes)

  For example, `BACKFILL_30` means streaming key mutations in the Couchbase server for five minutes and then stop stream.

The import command is as follows:

```
$bin/sqoop import -connect http://localhost:8091/pools -table DUMP
```

The preceding command dumps all the key values from the Couchbase server to HDFS.

If you are using multiple Couchbase servers, we can also specify the comma-separated list of the server in the `–connect` argument:

```
$bin/sqoop import-connect http://localhost:8091/pools,http://
localhost1:8091/pools -table DUMP
```

The export command is as follows:

```
$bin/sqoop export-connect http://localhost:8091/pools --tablegarbage_
value --export-dir /user/username/hdfsDir
```

The preceding command will export all the key-value pairs from the HDFS directory to the Couchbase server.

> The `–table` argument is required while exporting data from HDFS to Couchbase, but internally, the Couchbase plugin doesn't use this value.

## The Netezza Sqoop connector

The Netezza Sqoop connector allows importing data from Netezza to HDFS. We can download the Netezza Sqoop connector from `https://ccp.cloudera.com/display/con/Cloudera+Connector+for+Netezza`.

To install the Cloudera connector for Netezza, you need to extract the downloaded `tar.gz` file at some location on your machine. I am assuming you have extracted Netezza at `/opt/sqoop/plugin/Netezza/`. Now, copy the `sqoop-nz-connector-1.0.5.jar` file from `/opt/sqoop/plugin/Netezza/` to `$SQOOP_HOME/lib`.

Next we need to create the `connectors` file under `$SQOOP_HOME/conf/managers.d`. We also need to create the `managers.d` folder if it doesn't exist.

Add the following lines in the `connectors` file:

```
Com.cloudera.sqoop.manager.NetezzaManagerFactory=/usr/lib/sqoop-nz-
connector-1.0.5/sqoop-nz-connector-1.0.5.jar
```

The import command is as follows:

```
$ bin/sqoop import -connect jdbc:etezza://localhost/db1 -username user -
password password-direct -table tableName-num-mappers 4 -escaped-by '\\'
-fields-terminated-by ','
```

The preceding command will read data from Netezza and dump it into HDFS; `--direct` is used to support direct mode for fast access.

The export command is as follows:

```
$ bin/sqoop export -connect jdbc:etezza://localhost/db1 -username user
-password password -direct -export-dir /user/username/tableName -table
tableNameTarget -num-mappers 4 -input-escaped-by '\\'
```

The preceding command will read data from HDFS at `/user/username/tableName` and dump it into the table `NameTarget`.

## Microsoft SQL Server connector

The Microsoft SQL Server connector for Hadoop allows reading the data from the Microsoft SQL Server and dumping it into HDFS.

We can download the Microsoft SQL Server connector from the following location:

`http://www.microsoft.com/en-in/download/details.aspx?id=27584`

The preceding link also contains the installation guide, which we need to refer to for installation of the Microsoft SQL Server connector into Sqoop.

The following command imports data from RDBMS to HDFS:

```
$bin/sqoop import –connect 'jdbc:sqlserver://localhost;username=userna
me;password=password;database=db1' –table tableName–target-dir /user/
username/tableName
```

The previous command will read data from the table `tableName` and dump it into HDFS at `/user/username/tableName`.

The following command imports data from RDBMS to Hive:

```
$bin/sqoop import –connect 'jdbc:sqlserver://localhost;username=username;
password=password;database=db1' –table tableName–hive-import
```

The preceding command will transfer data from the RDBMS table to the Hive table.

The following command exports data from RDBMS to Hive:

```
$bin/sqoop export –connect 'jdbc:sqlserver://localhost;username=username
;password=password;database=db1' –table tableNameTarget  --export-dir /
user/username/tableName
```

The preceding command will read data from HDFS and dump it into the table `tableNameTarget` in the MS SQL Server.

## Quest Data connector

The Quest Data connector allows fast access of data from Oracle to HDFS. We can download the Quest Data connector for Hadoop and Oracle from the following location:

`https://ccp.cloudera.com/display/SUPPORT/Downloads`

The preceding link contains the installation guide, which we need to refer to for installation of the Quest Data connector into Sqoop.

The import command is as follows:

```
$ bin/sqoop import –connect jdbc:oracle:thin:@localhost:1521:db1 –
username username –password password–table tableName –target-dir /user/
username/tableName
```

The previous command will read data from the Oracle table `tableName` and dump it into HDFS at `/user/username/tableName`.

The export command is as follows:

```
$ bin/sqoop export --connect jdbc:oracle:thin:@localhost:1521:db1
--username username -password password --table tableName --export-dir /
user/username/tableName
```

The preceding command will read data from the HDFS directory and dump it into the Oracle table, `tableName`.

## Teradata Sqoop connector

The Teradata Sqoop connector allows importing data from Teradata to HDFS. We can download the Teradata Sqoop connector from `https://ccp.cloudera.com/display/SUPPORT/Downloads`.

To install the Cloudera connector for Teradata, you need to extract the downloaded `tar.gz` file at some location on your machine. I am assuming that you have extracted the Teradata at `/opt/sqoop/plugin/Teradata/` on your machine.

We also need to download the Teradata JDBC driver from the following location `http://downloads.teradata.com/download/connectivity/jdbc-driver` and then copy it into the `$SQOOP_HOME/lib` folder.

Next we need to create the `connectors` file inside the `$SQOOP_HOME/conf/managers.d` folder. We also need to create the `managers.d` folder if it doesn't exist.

Add the following lines in the `connectors` file:

```
com.cloudera.sqoop.manager.TeradataManagerFactory=/usr/lib/sqoop-td-connector-1.1.1/sqoop-td-connector-1.1.1.jar
```

If we don't want to create a new configuration file, we can add the following lines in the `sqoop-site.xml` file:

```
<configuration>
  <property>
    <name>sqoop.connection.factories</name>
    <value>com.cloudera.sqoop.manager.TeradataManagerFactory</value>
  </property>
</configuration>
```

The import command is as follows:

```
$ bin/sqoop import –connectjdbc:teradata://localhost/DATABASE=db1--
username username  --password password --table tableName --num-mappers 4
--escaped-by '\\' --fields-terminated-by ',' --lines-terminated-by '\n'
```

The previous command will read data from Teradata and dump it into HDFS at `/user/username/tableName`.

The export command is as follows:

```
$ bin/sqoop export --connect jdbc:teradata://localhost/DATABASE=db1
--username username --password password --export-dir /user/username/
tableName --table tableNameTarget --num-mappers 4 --input-escaped-by '\\'
\
--input-fields-terminated-by ','
```

The preceding command will read data from HDFS at `/user/username/tableName` and dump it into the table `tableNameTarget` in Teradata.