# **Getting Data into Hadoop**

<http://www.informit.com/articles/article.aspx?p=2756471&seqNum=4>

**Importing Data from Files into Hive Tables**

Apache Hive is an SQL-like tool for analyzing data in HDFS. Data scientists often want to import data into Hive from existing text-based files exported from spreadsheets or databases. These file formats often include tab-separated values (TSV), comma-separated values (CSV), raw text, JSON, and others. Having the data in Hive tables enables easy access to it for subsequent modeling steps, the most common of which is feature generation, which we discuss in Chapter 5, “Data Munging with Hadoop.”

Once data are imported and present as a Hive table, it is available for processing using a variety of tools including Hive’s SQL query processing, Pig, or Spark.

Hive supports two types of tables. The first type of table is an *internal table* and is fully managed by Hive. If you delete an internal table, both the definition in Hive *and* the data will be deleted. Internal tables are stored in an optimized format such as ORC and thus provide a performance benefit. The second type of table is an *external table* that is not managed by Hive. External tables use only a metadata description to access the data in its raw form. If you delete an external table, only the definition (metadata about the table) in Hive is deleted and the actual data remain intact. External tables are often used when the data resides outside of Hive (i.e., some other application is also using/creating/managing the files), or the original data need to remain in the underlying location even after the table is deleted.

Due to the large number of use cases, we do not cover all the input methods available to Hive, and instead just a basic example of CSV file import is described. Interested readers can consult the Hive project page, [https://hive.apache.org](https://hive.apache.org/), for more information.

**Import CSV Files into Hive Tables**

The following example illustrates how a comma delimited text file (CSV file) can be imported into a Hive table. The input file (names.csv) has five fields (Employee ID, First Name, Title, State, and type of Laptop). The first five lines of the file are as follows:

10,Andrew,Manager,DE,PC

11,Arun,Manager,NJ,PC

12,Harish,Sales,NJ,MAC

13,Robert,Manager,PA,MAC

14,Laura,Engineer,PA,MAC

The first input step is to create a directory in HDFS to hold the file. Note that, like most Hadoop tools, Hive input is directory-based. That is, input for an operation is taken as all files in a given directory. The following command creates a names directory in the users HDFS directory.

$ hdfs dfs -mkdir names

In this example, one file is used. However, any number of files could be placed in the input directory. Next the names.csv file is moved into the HDFS names directory.

$ hdfs dfs -put name.csv names

Once the file is in HDFS, we first load the data as an external Hive table. Start a Hive shell by typing hive at the command prompt and enter the following commands. Note, to cut down on clutter, some of the non-essential Hive output (run times, progress bars, etc.) have been removed from the Hive output.

hive> CREATE EXTERNAL TABLE IF NOT EXISTS Names\_text(

> EmployeeID INT,FirstName STRING, Title STRING,

> State STRING, Laptop STRING)

> COMMENT 'Employee Names'

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY ','

> STORED AS TEXTFILE

> LOCATION '/user/username/names';

OK

If the command worked, an OK will be printed. The various fields and the comma delimiter are declared in the command. The final LOCATION statement in the command tells Hive where to find the input files. The import can be verified by listing the first five rows in the table:

hive> Select \* from Names\_text limit 5;

OK

10 Andrew Manager DE PC

11 Arun Manager NJ PC

12 Harish Sales NJ MAC

13 Robert Manager PA MAC

14 Laura Engineer PA MAC

The next step is to move the external table to an internal Hive table. The internal table must be created using a similar command. However, the STORED AS format offers new options. There are four main file formats for Hive tables in addition to the basic text format. The choice of format depends on the type of data and analysis, but in most cases either ORC or Parquet are used as they provide the best compression and speed advantages for most data types.

Text file—All data are stored as raw text using the Unicode standard.

Sequence file—The data are stored as binary key/value pairs.

RCFile—All data are stored in a column optimized format (instead of row optimized).

ORC—An optimized row columnar format that can significantly improve Hive performance.

Parquet—A columnar format that provides portability to other Hadoop tools including Hive, Drill, Impala, Crunch, and Pig.

The following command creates an internal Hive table that uses the ORC format:

hive> CREATE TABLE IF NOT EXISTS Names(

> EmployeeID INT,FirstName STRING, Title STRING,

> State STRING, Laptop STRING)

> COMMENT 'Employee Names'

> STORED AS ORC;

OK

To create a table using one of the other formats, change the STORED AS command to reflect the new format. Once the table is created, the data from the external table can be moved to the internal table using the command,

hive> INSERT OVERWRITE TABLE Names SELECT \* FROM Names\_text;

As with the external table, the contents can be verified using the following command:

hive> Select \* from Names limit 5;

OK

10 Andrew Manager DE PC

11 Arun Manager NJ PC

12 Harish Sales NJ MAC

13 Robert Manager PA MAC

14 Laura Engineer PA MAC

Hive also supports partitions. With partitions, tables can be separated into logical parts that make it more efficient to query a portion of the data. For example, the internal Hive table created previously can also be created with a partition based on the state field. The following command creates a partitioned table:

hive> CREATE TABLE IF NOT EXISTS Names\_part(

> EmployeeID INT,

> FirstName STRING,

> Title STRING,

> Laptop STRING)

> COMMENT 'Employee names partitioned by state'

> PARTITIONED BY (State STRING)

> STORED AS ORC;

OK

To fill the internal table from the external table for those employed from PA, the following command can be used:

hive> INSERT INTO TABLE Names\_part PARTITION(state='PA')

> SELECT EmployeeID, FirstName, Title, Laptop FROM Names\_text WHERE state='PA';

...

OK

This method requires each partition key to be selected and loaded individually. When the number of potential partitions is large, this can make data entry inconvenient. To address this issue Hive now supports **dynamic-partition insert** (or multi-partition insert) that is designed to solve this problem by dynamically determining which partitions should be created and populated while scanning the input table.