The ROW FORMAT Clause

As you're probably aware, data files can come in different formats. For example, a file might be *comma-delimited,* which means the comma (,) is used to mark *(delimit)* when one column's value ends and the next column's value begins. The tab character is often used instead, giving a *tab-delimited* file.

As part of the **CREATE TABLE** statement, you can specify how the data is delimited in its files. This is done using the **ROW FORMAT** clause. The syntax for this clause is:

```
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY character
```

For example, consider this row from a data file:

```
1,Data Analyst,135000,2016-12-21 15:52:03
```

There are four fields here, separated by commas: an ID, a job title, a salary, and a timestamp when the data was recorded. The following statement will create the table:

```
CREATE TABLE jobs (id INT, title STRING, salary INT, posted TIMESTAMP)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ',';
```

The **ROW FORMAT DELIMITED** portion of this clause specifies *that* you are using a delimiter. You also need the **FIELDS TERMINATED BY** portion, to specify *which* delimiter you are using. In this case, the delimiter is the comma (,). If the delimiter were the tab character, the clause would use **t** in quotes (because **t** is the *escape sequence* representing the tab character):

```
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t'
```

The **FIELDS TERMINATED BY** portion is a part of the **ROW FORMAT** clause. It *must* be preceded by **ROW FORMAT DELIMITED**. The line break and indentation used in these examples is optional.

If you omit the **ROW FORMAT** clause, Hive and Impala will use the default field delimiter, which is the ASCII Control+A character. This is a non-printing character, so when you attempt to view this character using a text editor or a **cat** command, it might render as a symbol (such as a rectangle with 0s and 1s in it) and other characters might overlap with it.

Try It!

In the following exercises, you can see how the **ROW FORMAT** clause dictates storage of new data for a table, and how it tells Hive and Impala how to correctly read a table in existing data files.

On the VM:

1. Log in to Hue and go to the Impala query editor.

2. Do the following to create a comma-delimited table, fill it with one row of data, and look at the resulting file on HDFS:

   a. Execute the **CREATE TABLE** statement:

```
    CREATE TABLE jobs
        (id INT, title STRING, salary INT, posted TIMESTAMP)
        ROW FORMAT DELIMITED

        FIELDS TERMINATED BY ',';
```

b. Load one row of data by executing the following statement. (**Note:** This statement is *not* a good way to add a lot of data to a table in a big data system, for reasons described later in the course. Here, we're only adding one row for demonstration purposes.)

```
INSERT INTO jobs

    VALUES (1,'Data Analyst',135000,'2016-12-21 15:52:03');
```

c. Use the File Browser or the data source panel on the left side (choosing the files icon rather than the database icon) and find the **/user/hive/warehouse/jobs** directory. If you don't see the **jobs** subdirectory, refresh the display by clicking the refresh button (two curved arrows). Find a file with a name that's just a string of letters and numbers, and click that file.

d. You can see the contents of the file in the main panel. Notice that you have a comma-delimited row of data.

e. Notice that the string and timestamp values stored in this file are *not* enclosed in quotes. Quotes were used in the **INSERT** statement to enclose the literal string and timestamp values, but Hive and Impala do *not* store these quotation marks in the table's data files.

3. Now go back to the Impala query editor and create a tab-delimited table, fill it with the same data, and compare the resulting file on HDFS to what you had for the comma-delimited file:

a. Execute the **CREATE TABLE** statement (the only differences are the table name and the delimiting character):

```
CREATE TABLE jobs_tsv

        (id INT, title STRING, salary INT, posted TIMESTAMP)

    ROW FORMAT DELIMITED

        FIELDS TERMINATED BY '\t';
```

b. Load one row of data by executing the following statement. (**Note:** This statement is *not* a good way to add a lot of data to a table in a big data system, for reasons described later in the course. Here, we're only adding one row for demonstration purposes.)

```
INSERT INTO jobs_tsv

    VALUES (1,'Data Analyst',135000,'2016-12-21 15:52:03');
```

c. Use the File Browser or the data source panel on the left side (choosing the files icon rather than the database icon) and find the **/user/hive/warehouse/jobs_tsv** directory. If you don't see the **jobs_tsv** subdirectory, refresh the display by clicking the refresh button (two curved arrows). Find a file with a name that's just a string of letters and numbers, and click that file.

d. You can see the contents of the file in the main panel. Notice that this time, you have a *tab*-delimited row of data.

e. Notice that the string and timestamp values stored in this file are *not* enclosed in quotes. Quotes were used in the **INSERT** statement to enclose the literal string and timestamp values, but Hive and Impala do *not* store these quotation marks in the table's data files.

4. Drop the **jobs** and **jobs_tsv** tables. (Look back at the "Creating Databases and Tables…" readings for how to drop tables, if necessary.)

5. When you're creating a table for existing files, you'll want to specify how the file is already delimited. Do the following steps to see this at work.

a. First, examine the data in the **/user/hive/warehouse/investors** directory. This will be the default location for a table named **default.investors**. There is no table for this data yet, so you will create one. Notice that the file is *comma-delimited*.

b. Create an *externally managed* **investors** table using the following statement (which purposefully does *not* specify the delimiter). ***It's important to use the EXTERNAL keyword,*** so you can drop the table without deleting the data.

```
CREATE EXTERNAL TABLE default.investors

    (name STRING, amount INT, share DECIMAL(4,3));
```

c. Use Hue to look at the table. It only has a few rows, so you can use **SELECT * FROM investors;** in the query editor, or you can use the data source panel to view the sample data. Notice that the entire row ended up in the **name** column! This is because the command used the default delimiter, Control+A, rather than the comma.

d. Drop the table by entering and executing the command **DROP TABLE investors;** Check that the **/user/hive/warehouse/investors** directory still exists and has a file in it. (If you made a mistake and the directory is gone, see below!)

e. Now, create another **investors** table using the **ROW FORMAT** clause to specify the delimiter:

```
CREATE EXTERNAL TABLE default.investors

        (name STRING, amount INT, share DECIMAL(4,3))
```

```
   ROW FORMAT DELIMITED

       FIELDS TERMINATED BY ',';
```

f. Use Hue to look at the table. Now each column should have values. Keep this table, you will use it again later.

If you accidentally deleted your data:

Open a Terminal window. (You can do this by clicking the icon in the menu bar that looks like a computer.) Enter and run the following command (on one line), which will copy the file from your local disk to the proper place in HDFS. Do *not* include the **$**; that's the prompt to indicate this is a command-line shell command.

```
$ hdfs dfs -put ~/training_materials/analyst/scripts/static_data/default/investors  /user/hive/warehouse/
```