Using Hive SerDes

Traditional data processing relies on data being stored in structured tables with well-defined rows and columns. However, data is often not structured this way. A lot of data exists in *unstructured* or*semi-structured* text files, such as log files, free-form notes in electronic medical records, different types of electronic messages, and product reviews. This kind of data can provide valuable insights, but working with it requires a different approach.

Hive provides features for working with unstructured text data, semi-structured data in formats like JSON, and data that lacks consistent delimiters. Note that the features discussed in this reading are supported only by Hive, not by Impala.

Recall that the clause ROW FORMAT DELIMITED is used when creating a table with data stored in text files. When you create a table using this clause, the data in the underlying text files must be organized into rows and columns with consistent delimiters.

Hive SerDes

Hive provides interfaces called *SerDes* that can read and write data that is *not*in a structured tabular format. SerDe stands for *serializer/deserializer*. *Serializing* is the process for converting data to bytes so it can be stored; *deserializing* is the reverse process—decoding when reading the stored file.

You can specify a table's SerDe when you create the table. Actually, every table in Hive has a SerDe associated with it, whether you realize it or not. Typically, the SerDe is automatically set to the default for a given file format, based on the ROW FORMAT and STORED AS clauses.

For a table stored as text files, the default SerDe is called LazySimpleSerDe. The LazySimpleSerDe gets its name because it uses a technique called *lazy initialization* for better performance. That means it does not instantiate objects until they're needed.

In addition to LazySimpleSerDe, Hive includes several other built-in SerDes for working with data in text files. If you want to use one of these, you must explicitly specify the SerDe in the CREATE TABLE statement. The statement includes the clause ROW FORMAT SERDE followed by the fully qualified name of the Java class that implements the SerDe, enclosed in quotes.

For example, Hive includes the OpenCSVSerde, which can process data in comma-separated values (CSV) files. But why does Hive need a special CSV SerDe when the default SerDe can handle comma-delimited text files (when you specify ROW FORMAT DELIMITED FIELDS TERMINATED BY ',')? Although Hive's default SerDe can work with simple comma-delimited text data, the actual CSV format allows things like commas embedded within fields, quotes enclosing fields, and missing fields. Hive's default SerDe does not support these, but the OpenCSVSerde does. The OpenCSVSerde also supports other delimiters such as the tab and pipe characters.

The Try It! section of "The ROW FORMAT Clause" reading presented an example where you incorrectly create a table from a comma-delimited file, because the delimiter was not specified. You corrected this by using a ROW FORMAT DELIMITED clause, but you also could have corrected it using the OpenCSVSerde:

```
CREATE EXTERNAL TABLE default.investors

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

  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';
```

Two other examples are RegexSerDe, which identifies individual fields within each record based on a regular expression, and JsonSerDe, which processes data in JSON format. See the next reading for more on these SerDes.

In addition to using SerDes to read and write text data, Hive also uses SerDes to read to and write from binary and columnar formats like Avro and Parquet, but Hive does this automatically and hides the details from the user. Note, however, that the SerDe is not the same as the file type, but there is a close connection—file types require particular SerDes so Hive can read and write those file types. You'll learn more about file types in the next Week's materials. For now, just remember that SerDes define *processes* for reading data files.

While OpenCSVSerde allows you to both read and write files using the specified formatting, some SerDes will only read files; you cannot use them to write. You should test your SerDes or read the documentation to determine if writing files is supported.

Try It!

Do the following to create a table named tunnels in the dig database.

1. Examine the data in the file training_materials/analyst/data/tunnels.csv on the local file system of the VM. Note that it's a comma-delimited text file.

2. In the *Hive* query editor, execute the following statement. Note that it uses the OpenCSVSerde rather than the ROW FORMAT DELIMITED syntax you used in the previous lesson. You still could use that other syntax; this is meant to illustrate that OpenCSVSerde does the same thing. Also you *must* use Hive to execute the statement below, not Impala.

```
CREATE TABLE dig.tunnels

    (terminus_1 STRING, terminus_2 STRING, distance SMALLINT)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';
```

3. In a Terminal window, run the following statement (all on one line) to move the tunnels.csv into the table directory. Do not include the $; that's the prompt to indicate this is a command-line shell command, not a query.

```
$ hdfs dfs -put ~/training_materials/analyst/data/tunnels.csv /user/hive/warehouse/dig.db/tunnels/
```

You'll learn more about this statement later in this course.

4. Use the data source panel or a Hive SELECT * statement to verify that the tunnels table has the data, with values in the correct columns. Remember that you can query this table only with Hive, not with Impala.