Creating Tables with Complex Data

The syntax for creating tables that use complex data types is very similar in Hive and Impala, but mostly you will be creating tables in Hive. The reason for that is *Impala only supports the use of complex data in Parquet files*, and you cannot load complex data into a table using INSERT or LOAD statements in Impala. If you don't have the data file in Parquet format, you can create the table in Hive, then create a copy using CREATE TABLE … AS SELECT, with STORED AS PARQUET. You then can query the table in Impala.

The examples below assume you are using text files to store the data, so the delimiters are specified in the ROW FORMAT clause of the CREATE TABLE statement. For file formats such as Parquet and Avro, you do not use the ROW FORMAT clause; the details of how these formats represent complex values are determined by the file format, not by the user.

If you're using Impala to create the tables, you *must* be using Parquet files, so Impala will never use the ROW FORMAT clause for tables with complex data, and you will need to specify STORED AS PARQUET. The CREATE TABLE statements otherwise will be the same as the examples shown here.

(Querying data can be very different, though, so the next two readings will cover basic queries in both engines, one at a time.)

ARRAY

An ARRAY type is declared in the column list of the CREATE TABLE statement using ARRAY< *type*>, where *type* is the simple data type that each element of the array will have.

The following shows the contents of a data file called customers_phones_array.csv with three columns: cust_id, name, and phones. This will be used as the data for a table using an ARRAY data type for phones using ARRAY<STRING> because the phone numbers are given as STRINGs.

```
a,Alice,555-1111|555-2222|555-3333

b,Bob,555-4444

c,Carlos,555-5555|555-6666
```

Commas separate the customer ID, the customer name, and a list of their phone numbers. The phone numbers themselves are separated using the pipe character (the vertical bar). Both delimiters need to be declared in the CREATE TABLE statement:

```
CREATE TABLE customers_phones_array

    (cust_id STRING,

    name STRING,

    phones ARRAY<STRING>)

  ROW FORMAT DELIMITED

    FIELDS TERMINATED BY ','

    COLLECTION ITEMS TERMINATED BY '|';
```

Recall that if you omit the FIELDS TERMINATED BY subclause, Hive and Impala use the default delimiter, which is the ASCII Control-A character. For the COLLECTION ITEMS TERMINATED BY subclause, the default collection item terminator is the ASCII Control-B character.

Remember: If you're creating the table using Parquet or Avro data files (and, again, Parquet is the only format Impala supports with complex data) omit the ROW FORMAT clause and the subclauses specifying the terminators, and include a STORED AS clause.

MAP

A MAP type is declared in the column list of the CREATE TABLE statement using MAP<*keytype, valuetype*>. Notice that the keys—in the phones example, this would be home, work, and mobile—are not defined in the CREATE TABLE statement. This means new keys could be added to the data without updating the table definition.

The following shows the contents of a data file called customers_phones_map.csv with the same three columns as in the ARRAY example: cust_id, name, and phones. In this case, though, phones will use MAP<STRING,STRING> because both the key (type of number) and the value (the phone number itself) are both given as STRINGs.

```
a,Alice,home:555-1111|work:555-2222|mobile:555-3333

b,Bob,mobile:555-4444

c,Carlos,work:555-5555|home:555-6666
```

Again, as with the ARRAY example, commas separate the customer ID, the customer name, and the list of their phone numbers. The key-value pairs are separated using the pipe character (the vertical bar) again, and colons are used to separate the key from the value in each pair. All three delimiters need to be declared in the CREATE TABLE statement:

```
CREATE TABLE customers_phones_map

    (cust_id STRING,

    name STRING,

    phones MAP<STRING,STRING>)

  ROW FORMAT DELIMITED
```

FIELDS TERMINATED BY ','

COLLECTION ITEMS TERMINATED BY '|'

MAP KEYS TERMINATED BY ':';

The default terminators are the same as with ARRAY, but now you also have the MAP KEYS terminator. The default (if you omit the MAP KEYS TERMINATED BY subclause) is the ASCII Control-C character.

Remember: If you're creating the table using Parquet or Avro data files (and, again, Parquet is the only format Impala supports with complex data) omit the ROW FORMAT clause and the subclauses specifying the terminators, and include a STORED AS clause.

STRUCT

A STRUCT type is declared in the column list of the CREATE TABLE statement using STRUCT<*field1:TYPE1, field2:TYPE, …*>. The order of the STRUCT fields in the table definition *must* match the order in the data files.

The following shows the contents of a data file called customers_addr.csv with the three columns: cust_id, name, and address. Here, address will use a STRUCT type with four named fields: street, city, state, and zipcode. All are STRINGs except zipcode, which is an INT.

a,Alice,742 Evergreen Terrace|Springfield|OR|97477

b,Bob,1600 Pennsylvania Ave NW|Washington|DC|20500

c,Carlos,342 Gravelpit Terrace|Bedrock

A STRUCT contains a predefined number of named fields, but fields can be missing. In this example, Carlos’s address is missing the state and zipcode fields, so queries will return NULL for these missing fields.

Again, as with the ARRAY example, commas separate the columns. The fields in the STRUCT are separated using the pipe character (the vertical bar). Both delimiters need to be declared in the CREATE TABLE statement:

```
CREATE TABLE customers_addr
        (cust_id STRING,
        name STRING,
        address STRUCT<street:STRING,
                       city:STRING,
                       state:STRING,
                       zipcode:INT>)
    ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      COLLECTION ITEMS TERMINATED BY '|';
```

The default terminators are the same as with ARRAY.

Note that unlike MAPs, the “keys” of a STRUCT (the names of its fields) are *not* part of the actual data. So if we changed the name from zipcode to postalcode, we would *not* need to update the underlying data in the data file.

Remember: If you're creating the table using Parquet or Avro data files (and, again, Parquet is the only format Impala supports with complex data) omit the ROW FORMAT clause and the subclauses specifying the terminators, and include a STORED AS clause.

Try It!

Do the following to create tables that you can query in the next readings.

1. Use Hive to create the example tables for each of the three types, and load the data with your preferred method. The data for each example is on the VM in /home/training/training_materials/analyst/data/. The data files are named customers_phones_array.csv, customers_phones_map.csv, and customers_addr.csv.
2. Use Hive to create a Parquet version of each table so you can also query the data with Impala. For each table, run a CTAS statement and use STORED AS PARQUET. To make things easier when you query these tables with Impala, name the tables phones_array_parquet, phones_map_parquet, and customers_addr_parquet. If you prefer to use something shorter, please do, but you'll need to make adjustments when you complete the exercises in the “Querying Complex Data with Impala” reading.

For example:

```
CREATE TABLE phones_array_parquet
    STORED AS PARQUET
    AS SELECT * FROM customers_phones_array;
```