Complex Data Types

Recall that Hive and Impala support a number of simple data types, similar to the types found in relational databases. These simple data types represent a single value within a single row-column position.

In addition, Hive and Impala also support several *complex data types*, which represent multiple values within a single row-column position. Complex types are also referred to by several other names, including *nested types* and *collection types*.

Hive and Impala both support three different complex data types: ARRAY, MAP, and STRUCT.

ARRAY

An ARRAY represents an ordered list of values, all having the same data type. For example, people often have multiple phone numbers, such as home (landline), work, and mobile. An array could hold several phone numbers. In the table below, the column phones is an ARRAY in which each element is a STRING:

name	phones
Alice	[555-1111, 555-2222, 555-3333]
Bob	[555-4444]
Carlos	[555-5555, 555-6666]

The elements of an ARRAY can be other simple data types, but all elements of an ARRAY must be of the same type.

MAP

A MAP represents key-value pairs, with all keys having the same data type, and all values having the same type. With the phones example, this allows you to specify which phone number is for what purpose (such as home, work, or mobile):

name	phones
Alice	{home:555-1111, work:555-2222, mobile:555-3333}
Bob	{mobile:555-4444}
Carlos	{work:555-5555, home:555-6666}

Here the key is a STRING and the value is also a STRING. Each could be other simple data types; for example, if you don't use the dash in the phone numbers, you could make the key STRING and the value INT.

STRUCT

A STRUCT represents named fields, which can have different data types. For example, you could use a STRUCT to store addresses, with each part of the address a different field:

name	address
Alice	{street:742 Evergreen Terrace, city:Springfield, state:OR, zipcode:97477}
Bob	{street:1600 Pennsylvania Ave NW, city:Washington, state:DC, zipcode:20500}
Carlos	{street:342 Gravelpit Terrace, city:Bedrock}

Here, the STRUCT is defined to have four fields: street, city, and state are STRING types; zipcode is an INT type (though it could also be STRING). Notice that Carlos's address is missing the state and zipcode fields. When this table is queried (see the next readings), those fields would show as NULL.

Nested Complex Types

It's also possible to *nest* complex types, for example, to have an ARRAY in which each element is an ARRAY, or a MAP for which the value is a STRUCT element. For the tables above, you might create a contacts column which is a STRUCT with two named fields: phones is a MAP and address is another STRUCT.

As you'll see in the next readings, working with a single layer of complex data can be difficult; working with a nested layer will be much more difficult. If you do need to use nested complex types, we recommend using no more than one nested layer. If you find yourself using more (for example, an ARRAY whose elements are MAPs, and the values of that MAP are themselves ARRAYs), consider whether a different schema design could provide the same information in a more digestible way.