**DATATYPES IN CQL (2.0)**

**CUSTOM DATA TYPE :**

A Custom datatype is defined by every Java class that extends the server-side AbstractType class and can be loaded by Cassandra is called a custom type; as a result, it should be included in the CLASSPATH of every node that is running Cassandra. When used as a clustering column, that class will specify which values are appropriate for the type and how the time sorting works. A custom type's value can be entered using the blob literal syntax and is equivalent to a blob for all other purposes.

When hex is a hexadecimal character, as [0-9a-fA-F], the Cassandra blob data type is used to represent a constant hexadecimal number, which is defined as 0[xX](hex)+. As an instance, 0xcafe. A blob can have a maximum size of 2 GB in theory. However, blob size is practically limited to less than 1 MB. A brief string or tiny image can be stored in a blob type.

These functions convert the native types into binary data (blob):

* typeAsBlob(value)
* blobAsType(value)

The typeAsBlob function accepts an argument of any native, nonblob data type that CQL supports and returns it as a blob. On the other hand, if feasible, the blobAsType function transforms a 64-bit blob input into a value of the designated data type.

Here's an example of using bigintAsBlob:

CREATE TABLE bios ( user\_name varchar PRIMARY KEY,

bio blob

);

INSERT INTO bios (user\_name, bio) VALUES ('fred', bigintAsBlob(3));

SELECT \* FROM bios;

user\_name | bio

-----------+--------------------

fred | 0x0000000000000003

**TUPLES DATA TYPE :**

Sets of typed positional fields with a fixed length are stored in tuples. Instead of using a user-defined type, use a tuple. There are more fields (32768) that a tuple can have than can be wisely employed. Usually, make a tuple consisting of a few fields.

Tuples and tuple types (where the elements can be of different kinds) are likewise supported by CQL. Tuples can be conceptualized functionally as anonymous UDTs with anonymous fields. The following define tuple types and tuple literals:

tuple\_type::= TUPLE '<' cql\_type( ',' cql\_type)\* '>'

tuple\_literal::= '(' term( ',' term )\* ')'

To declare the tuple component types in the table construction statement, use angle brackets and a comma as the delimiter. To insert tuple values into a table, enclose them in parenthesis, as demonstrated below:

CREATE TABLE collect\_things (

k int PRIMARY KEY,

v tuple<int, text, float>

);

INSERT INTO collect\_things (k, v) VALUES(0, (3, 'bar', 2.1));

SELECT \* FROM collect\_things;

k | v

---+-----------------

0 | (3, 'bar', 2.1)

A tuple is always frozen <frozen> (without requiring the frozen keyword), unlike other composite types like collections and UDTs, and it is not feasible to edit a tuple's partial members without also updating the entire tuple. Additionally, a tuple literal must always include the same number of values as those declared in the type of which it is a tuple (some of those values may be null, but they must be identified as so explicitly).