**COLLECTION DATATYPES IN CQL**

Collections are designed to store and denormalize modest amounts of data. They are useful for things like "labels applied to an email," "the phone numbers of a given user," and so on. However, collections are inappropriate in situations when items are expected to expand unbounded (e.g., "all messages sent by a user," "events registered by a sensor"). Instead, a special table containing clustering columns should be used. In particular, (non-frozen) collections have the following notable traits and restrictions:

* Internal indexing is not used for individual collections. This implies that a collection must be read in order to access even a single element (reading one is not paged internally).
* Certain operations on lists cause an internal read-before-write, although insertion operations on sets and maps never do. Furthermore, it can be challenging to retry some list operations in the event of a timeout because they are not idempotent by nature (for more information, see the section on lists below). Thus, if possible, it is recommended to favor sets over lists.

Please be aware that using a (single) collection to store a lot of data is not recommended, even though some of those restrictions might be changed or eliminated in the future.



**SET :**

A set is a collection of data that allows us to store a group of components together so that a user can have several email addresses. We use the set collection data type to store this kind of data, and it returns sorted elements in response to queries. It is an alphabetized list of distinct values.

SYNTAX :

Create table keyspace-Name.Table-Name(

 field 1 data\_type 1,

 /\*SET keyword is used to define collection data type.\*/

 field 1 set<data\_type>,

 field 1 data\_type 1,

 key\_constraint if any

 );

Creation of keyspace in CQL query :

CREATE KEYSPACE keyspace1

 WITH replication = {'class': 'SimpleStrategy',

 'replication\_factor' : 1};

Syntax for Insertion of data using set data type :

INSERT INTO table\_name(field 1, field 2, field 3..)

 VALUES('field1\_value', {'field2\_value1', 'field2\_value2', ..},

 { 'field3\_value1', 'field3\_value2', ..}....);

LIST :

One value may be stored in a list more than once. One of the constraints of the list collection data type is that components cannot be modified in order. entries in a list can be retrieved using specific indices that are assigned to each element once entries are saved. An element in a list does not have to be unique; it can be replicated.

Creation of table using CQL :

CREATE TABLE keyspace1.datafield

 (

 E\_id int,

 E\_name text,

 /\* list syntax to define Feild as list collection data type. \*/

 data list<text>,

 PRIMARY KEY(E\_id)

 );

SYNTAX FOR LIST DATA TYPE

INSERT INTO table\_name(field 1, field 2, field 3..)

 VALUES(' field1\_value', ['field2\_value1', 'field2\_value2', ..],

 [ 'field3\_value1', 'field3\_value2', ..]....);

MAP :

Typed collection of key-value pairs in MAP where each pair of values has a key name associated with it. Data in MAP collection is arranged according to distinct keys.

Creation of Table using CQL :

CREATE TABLE keyspace1.Activity

 (

 data int,

 data\_name text,

 task map<timestamp, text>,

 PRIMARY KEY(data)

 );

SYNTAX for map datatype

INSERT INTO table\_name(field 1, field 2, field 3..)

 VALUES({'key1':'value1', 'key2':'value2'..........});