Chapter 37.

March 2020

Welcome to the March 2020 edition of DataStax Developer's Notebook (DDN). This month we answer the following question(s);

> As a database application developer, I've never previously used a system with a natively asynchronous client side driver. What do I need to know; Can you help ?
>
> *Excellent question ! Yes, the DataStax Enterprise (DSE) client side drivers offer entirely native asynchronous operation.; fire and forget, or fire and listen. There are easy means to make the driver and any calls you issue block, and behave synchronously, but there's little fun in that.*
>
> *The on line documentation covers the asynchronous query topic well, so we'll review that and then extend into asynchronous write programming.*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.8 EAP (Early Access Program). All of the steps outlined below can be run on one laptop with 16 GB of RAM, or if you prefer, run these steps on Amazon Web Services (AWS), Microsoft Azure, or similar, to allow yourself a bit more resource.

For isolation and (simplicity), we develop and test all systems inside virtual machines using a hypervisor (Oracle Virtual Box, VMWare Fusion version 8.5, or similar). The guest operating system we use is Ubuntu Desktop version 18.04, 64 bit.

37.1 Terms and core concepts

As stated above, ultimately the end goal is to better understand the asynchronous abilities of the DataStax Enterprise (DSE) client side driver. Everything discussed below operates in an identical manner when using the open source Cassandra product as well.

The on line documentation for asynchronous operations, specifically reading, as really good and available here,

```
https://docs.datastax.com/en/developer/python-driver/3.21/api/cassandra/cluster/

https://github.com/datastax/python-driver/blob/master/examples/request_init_listener.py

https://docs.datastax.com/en/developer/python-dse-driver/2.11/api/dse/cluster/#dse.cluster.ResultSet
```

We use and list documentation for Python above, however; all of the DSE client side drivers work and support consistent features across languages. We use Python because it requires less overall code than JAva, other.

If we want to execute (and test) asynchronous operations, we'd best be served with something that will fail. An easy operation that fails when using DSE is to INSERT or similar a new data record with an incomplete primary key, specifically, an incomplete partition key. Example 37-1 displays the table we will use moving forward. A code review follow.

Example 37-1 Table we use in the examples that follow.

```
DROP KEYSPACE IF EXISTS ks_39;

CREATE KEYSPACE ks_39
   WITH replication = {'class': 'SimpleStrategy',
      'replication_factor': 1}
   AND graph_engine = 'Core';

USE ks_39;

CREATE TABLE t1
   (
   col1              INT,
   col2              INT,
   col3              INT,
   col4              INT,
   col5              INT,
```



```
   col6               INT,
   PRIMARY KEY((col1, col2), col3, col4)
   )
WITH VERTEX LABEL t1
;

INSERT INTO t1 (col1, col2, col3, col4, col5, col6)
   VALUES (1, 1, 1, 1, 1, 1);

//  These fail
//
// INSERT INTO t1 (col1, col3, col4, col5, col6)
//    VALUES (2, 2, 2, 2, 2);
// INSERT INTO t1 (col1, col2, col3, col4, col5, col6)
//    VALUES (3, null, 3, 3, 3, 3);

//  This works
INSERT INTO t1 (col1, col2, col3, col4, col5, col6) VALUES (4, 4, 4, 4, null,
4);
```

Relative to Example 37-1, the following is offered:

- We make the keyspace and table with support for graph, although none is required.
- The table, titled t1, has 6 columns, all integer. Columns col1 and col2 are part of the partition key, and may not be NULL at any time.
- NULL in Python is represented by the keyword, None.
- The first INSERT works, with all columns having non-NULL values.
- The second and third INSERTs fail, since the net effect is the col2, a member of the partition key, would be set to NULL.
- The last INSERT works.

The first part of our sample Python client side program is listed below in Example 37-2. A code review follows.

Example 37-2 Client side program example, connecting to the server

```
from dse.cluster import Cluster


l_cluster = Cluster(
   contact_points=['127.0.0.1']
```



```
   )
l_session = l_cluster.connect()

#  DSE; INSERT of NULL into partiton key generates error
#
#  NULL in Python is 'None'

l_stmt =                                                          \
   "INSERT INTO ks_39.t1 (col1, col2, col3, col4, col5, col6) " + \
   "   VALUES (%s, %s, %s, %s, %s, %s);                          "
```

Relative to Example 37-2, the following offered:

- This is pretty much the simplest (get started) Python client side program you can have; initialize a client side session with DataStax Enterprise (DSE).
- We're not using prepared statements, although we should for efficiency; didn't want to list the added complexity.

Example 37-3 lists our first example. A code review follows.

Example 37-3 First block of sample code

```
#  Block 1; synchronous INSERT loop


for i in range(12):
   print i
      #
   if (i == 8):
      l_resu = l_session.execute(l_stmt, ( i, None, i, i, i, i ) )
   else:
      l_resu = l_session.execute(l_stmt, ( i,    i, i, i, i, i ) )


#  Generates this error,
#
#   Traceback (most recent call last):
#     File "zzz.py", line 22, in <module>
#       l_resu = l_session.execute(l_stmt, ( i, None, i, i, i, i ) )
#     File "dse/cluster.py", line 2169, in dse.cluster.Session.execute
#     File "dse/cluster.py", line 4330, in dse.cluster.ResponseFuture.result
#   dse.InvalidRequest: Error from server: code=2200 [Invalid query]
message="Invalid null value in condition for column col2"
```



```
#  Without a try/except block above, this program terminates
```

Relative to Example 37-3, the following is offered:

- We loop for zero to 11, and perform a DSE INSERT. A branching is run when the loop counts to 8, and we run a different INSERT, one which attempts to INSERT a NULL column value for a column which is a member of this table's partition key.
- With no error handling, the program fails at this point, and abnormally terminates.
- The execute() method executes synchronously, so we will INSERT the values zero though seven, and no more; the program terminates.

Example 37-4 takes the example farther. A code review follows.

Example 37-4 Second block of our sample code

```
#  Block 2

#  This block differs from above in that the execute is now async-


for i in range(12):
   print i
      #
   if (i == 8):
      l_resu = l_session.execute_async(l_stmt, ( i, None, i, i, i, i ) )
   else:
      l_resu = l_session.execute_async(l_stmt, ( i,    i, i, i, i, i ) )


#  No error is generated, but row 8 is not inserted
```

Relative to Example 37-4, the follow is offered:

- In this example, we change from the execute() method to the execute_async() method.
- Rows in the range zero through 12 are INSERTed, minus the row valued 8. The program does not terminate, nor throw the error, since we aren't programming to observe or catch the error.

Example 37-5 continues the example. A code review follows.

Example 37-5 Third block of our sample code

```
#  Block 3

#  Add error handling around INSERT loop
#
#     This is not really what we want since result() is blocking.


for i in range(12):
   print i
      #
   if (i == 8):
      l_resu = l_session.execute_async(l_stmt, ( i, None, i, i, i, i ) )
      try:
         l_resu2 = l_resu.result()
      except Exception:
         print "ERROR 1: " + str(l_resu2)
   else:
      l_resu = l_session.execute_async(l_stmt, ( i,    i, i, i, i, i ) )
      try:
         l_resu2 = l_resu.result()
      except Exception:
         print "ERROR 2: " + str(l_resu2)


#  We catch the error; the program inserts all but 8. Output
#  as shown below (truncated)
#
#   ...
#   7
#   8
#   ERROR 1: <dse.cluster.ResultSet object at 0x7fe01a841690>
#   9
#   10
#   #   ...
```

Relative to Example 37-5, the following is offered:

- This is not the final code we want. By placing a result() method after the execute_async(), we are blocking, loosing the asynchronous processing we sought.
- We have more/better code below that (outputs) more from the results object; this topic is not expanded upon further here.

Example 37-6 captures results in a less blocking manner. A code review follows.

Example 37-6 Fourth block of our sample code

```
#  Block 4

#  Modified from above; we run the sync/result() after we run
#  all of the async executions
#
#  We don't like this one; error message isn't correct


#   l_resuList = []

#   for i in range(12):
#      print i
#         #
#      if (i == 8):
#         l_resuList.append(l_session.execute_async(l_stmt, ( i, None, i, i, i,
i ) ) )
#      else:
#         l_resuList.append(l_session.execute_async(l_stmt, ( i,    i, i, i, i,
i ) ) )

#   for l_resu in l_resuList:
#      try:
#         l_err = l_resu.result()
#      except Exception:
#         print "FAIL: " + str(l_err.__dict__)


#  Object returned above is ; dse.cluster.ResultSet
#
#  FAIL: {'column_names': None, '_list_mode': False, 'column_types': None,
#     '_page_iter': None, 'response_future': <ResponseFuture: query=
#        '<SimpleStatement query="INSERT INTO ks_39.t1 (col1, col2, col3,
#        col4, col5, col6)    VALUES (%s, %s, %s, %s, %s, %s); ",
#        consistency=Not Set>' request_id=8 result=None exception=None
#        coordinator_host=127.0.0.1:9042>, '_current_rows': []}
```

Relative to Example 37-6, the following is offered:

- Here we capture the handle to the result set inside a Python list, then execute the blocking, result() method.

- Besides blocking, we less than prefer the error message we have retrieved here.

Example 37-7 display the nearly last example code fragment. A code review follows.

Example 37-7 Fifth block of our sample code

```
#  Block 5

#  This is the one we like; just one event, before making it more
#  complex with all events from the INSERT loop


def my_okay(l_okay, level='debug'):
   print "OKAY: " + str(l_okay)
def my_fail(l_fail, l_stmt):
   print "FAIL: " + str(l_fail)


l_resu = l_session.execute_async(l_stmt, ( 20, None, 20, 20, 20, 20 ) )

l_resu.add_callbacks(callback=my_okay, callback_kwargs={'level': 'info'},
   errback=my_fail, errback_args=(l_stmt,))

#   FAIL: Error from server: code=2200 [Invalid query] message="Invalid null
value in condition for column col2"
```

Relative to Example 37-7, the following is offered:

- Almost done. This example has everything we want. We presented this simpler example, than the one below. The example below uses a Python List to maintain handles to an (array) of events from the INSERT loop.
- We define two functions, which are then registered as being our (futures/response listeners).

  If you get the function signature wrong between the function definition and later reference, your code will not work as desired, but wont error either.
- This INSERT will fail because of the NULL. And an error message we can usefully parse.

Example 37-8 displays the last code fragment. A code review follows.

Example 37-8 Sixth block of our sample code

```
#  Block 6

#  Enhancement over 5; actually perform the INSERT loop


def my_okay(l_okay, level='debug'):
   print "OKAY: " + str(l_okay)
def my_fail(l_fail, l_stmt, l_i):
   print "FAIL: " + str(l_fail) + " ... " + str(l_stmt) + " xxx " + str(l_i)

l_resuList = []

for i in range(12):
   print i
      #
   if (i == 8):
     l_resuList.append(l_session.execute_async(l_stmt, ( i, None, i, i, i, i )
) )
   else:
     l_resuList.append(l_session.execute_async(l_stmt, ( i,    i, i, i, i, i )
) )

   l_resuList[-1].add_callbacks(callback=my_okay, callback_kwargs={'level':
'info'},
      errback=my_fail, errback_args=(l_stmt, i, ))

# Full output
#
#   0
#   1
#   2
#   3
#   4
#   OKAY: None
#   5
#   OKAY: None
#   6
#   OKAY: None
#   7
#   OKAY: None
#   8
#   9
#   10
#   11
#   OKAY: None
#   FAIL: Error from server: code=2200 [Invalid query] message=
#      "Invalid null value in condition for column col2" ... INSERT
```



```
#      INTO ks_39.t1 (col1, col2, col3, col4, col5, col6)
#      VALUES (%s, %s, %s, %s, %s, %s);        xxx 8
#   OKAY: None
#   OKAY: None
#   OKAY: None
#   OKAY: None
#   OKAY: None
#   OKAY: None
```

Relative to Example 37-8, the following is offered:

- Again, we define two Python functions which will be invoked; one for success, one for fail.

  The argument lists between the function definition and the add_callbacks() method do not match, this code will not error, and also will not (catch) the response.

- l_resuList is a Python List, and maintains our handles to the (futures), the individual execute_async() statements.

- After the execute_async(), we register the call back functions. -1 is Python's way of referring to the last, most recently added element to the Python List; saves math.

- Output as shown.

  Notice we get the standard output, then, later, the notice of the async call being completed.

  Notice the helpful error information we receive.

37.2 Complete the following

At this point in this document we detailed how to INSERT. Recall that DataStax Enterprise INSERTs and UPDATEs are almost identical, each being converted to UPSERTs as needed. We didn't detail DELETES; perhaps experiment with those now ?

And the existing on line documentation covered queries really well.

37.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- A decent coverage of using (futures/callbacks) when running asynchronous DDL operations (INSERT, other) when using DataStax Enterprise (DSE) from the client side driver.
- We also covered some of the synchronous method calls.

Persons who help this month.

Kiyu Gabriel, Dave Bechberger, and Jim Hatcher.

Additional resources:

Free DataStax Enterprise training courses,

`https://academy.datastax.com/courses/`

Take any class, any time, for free. If you complete every class on DataStax Academy, you will actually have achieved a pretty good mastery of DataStax Enterprise, Apache Spark, Apache Solr, Apache TinkerPop, and even some programming.

This document is located here,

`https://github.com/farrell0/DataStax-Developers-Notebook`

`https://tinyurl.com/ddn3000`