```
%sh
STATUS="$(service cassandra status)"

if [[ $STATUS == *"is running"* ]]; then
    echo "Cassandra is running"
else
    echo " Cassandra not running .... Starting"
    service cassandra restart > /dev/null 2>&1 &
    echo " Started"
fi
```

READY

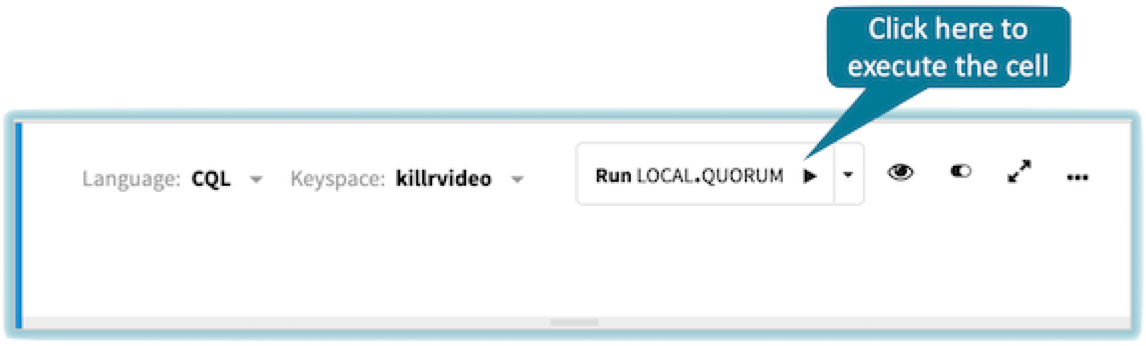
READY

Set Up the Notebook

In this section, you will do the following things:

- Execute a CQL script to initialize the KillrVideo database for this notebook

Step 1: Execute the following cell to initialize this notebook. Hover over the right-hand corner of the cell and click the *Run* button.



Language: CQL Keyspace: killrvideo Run LOCAL.QUORUM

Note: You don't see the CQL script because the code editor is hidden, but you can still run the cell.

READY

FINISHED

Collection Types

In this section, you will do the following things:

- Investigate SET , which is one of the collection types
- Insert and retrieve rows in the videos table that use SET

5_Advanced_Data_Types

The `videos` table uses a `SET` collection to keep track of tags associated with each video. A `SET` is a great collection to use because sets do not maintain an order - we are not concerned with any tag order, only if a tag is or is not associated with the video.

Let's start by reviewing the definition of the `videos` table:

Step 1: Execute the following cell to describe the `videos` table.

Took 0 sec. Last updated by anonymous at July 02 2020, 12:20:32 PM.

READY

```
// Execute this cell (click the Run button in the top-right corner)

DESCRIBE TABLE killrvideo.videos;
```

READY

Note two things about the `videos` table. First, the primary key is just `videoid`. Second, the `tags` column is a set of text. Tags are words or phrases we want to associate with a video.

To allow us to keep our focus on `SET`, in this example we will only specify the `videoid` and the `tags`. Once again, let's use our contrived `uuid` of `12121212-1212-1212-1212-121212121212`.

Step 2: In the following cell, insert a sparse row into the videos table with a `videoid` of `12121212-1212-1212-1212-121212121212` and a set of tags that contain the words: `Favorite`, `Fast-paced`, `Funny`.

Need a hint? Click here.

You want to `INSERT` into the `killrvideo.videos` table with a `videoid` of `12121212-1212-1212-1212-121212121212` and a set of tags such as `{ 'Favorite', 'Fast-paced', 'Funny' }`.\

Want the command? Click here.

```
INSERT INTO killrvideo.videos (videoid, tags)
  VALUES(12121212-1212-1212-1212-121212121212, { 'Favorite', 'Fast-paced', 'Funny' })
```

READY

```
// Write a command to insert a row into the videos table
// Execute this cell (click the Run button in the top-right corner)
```

READY

Now, let's check to see if our insert worked as expected.

Step 3: Execute the following cell to query for the row with the `videoid` of `12121212-1212-1212-1212-121212121212`.

```
// Execute this cell (click the Run button in the top-right corner)
SELECT * FROM killrvideo.videos WHERE videoid = 12121212-1212-1212-1212-121212121212;
```

READY

Inspect the `tags` values and see that the `INSERT` worked as expected.

READY

There are two kinds of `SET` updates we could perform. We can completely replace a set, or we can modify the contents of an existing set. First, we'll replace the entire `tags` set with the values `High-brow`, `Intellectual` and `Refined`.

Step 4: In the following cell, write a comand to replace the `tags` set for the `videoid` of `12121212-1212-1212-1212-121212121212`.

Need a hint? Click here.

You want to `UPDATE` the `killrvideo.videos` table with a `videoid` of `12121212-1212-1212-1212-121212121212`. `SET` the `tags` value to the new set `{ 'High-brow', 'Intellectual', 'Refined' }`.\

Want the command? Click here.

```
UPDATE killrvideo.videos SET tags = { 'High-brow', 'Intellectual', 'Refined' } WHERE
```

READY

```
// Write a command to update the row from the videos table
// Execute this cell (click the Run button in the top-right corner)
```

READY

Once again, let's inspect the effect of the `UPDATE`.

Step 5: Execute the following cell - a query to retrieve the row for the `videoid` of `12121212-1212-1212-1212-121212121212`.

READY

5_Advanced_Data_Types

```
// Execute this cell (click the Run button in the top-right corner)
SELECT * FROM killrvideo.videos WHERE videoid = 12121212-1212-1212-1212-121212121212;
```

READY

READY

We see the values we updated in Step 3. The values may not be in the same order as in your `UPDATE` command, but that's OK.

Thought question: If you *were* concerned about the order of the tags, what data type would you use instead of a `SET`?

Let's modify the set again. This time we will remove the `Refined` tag. Then in later steps we will replace it with `Low-rent`.

Step 6: In the following cell, write a command to update, by removing the `Refined` tag, for the `videoid` of `12121212-1212-1212-1212-121212121212`.

Need a hint? Click here.

Here, you will use an `UPDATE` command. Again, we are updating the row in the `killrvideo.videos` table with the `videoid` of `12121212-1212-1212-1212-121212121212`. The clause you use to remove the tag looks like `tags = tags - { 'Refined' }`.\

Want the command? Click here.

```
UPDATE killrvideo.videos SET tags = tags - { 'Refined' } WHERE videoid = 12121212-121
```

```
// Write a command to update the row from the videos table
// Execute this cell (click the Run button in the top-right corner)
```

READY

Again, let's check the contents of the row to see the effects of our command.

READY

Step 7: Execute the following cell - a query to retrieve the row for the `videoid` of `12121212-1212-1212-1212-121212121212`.

```
// Execute this cell (click the Run button in the top-right corner)

SELECT * FROM killrvideo.videos WHERE videoid = 12121212-1212-1212-1212-121212121212;
```

READY

Inspecting the previous results, we see the row now only has two tags - that's what we wanted!

READY

5_Advanced_Data_Types

Let's add a third tag `Low-rent`.

Step 8: In the following cell, write a command to update, by adding the `Low-rent` tag, for the `videoid` of `12121212-1212-1212-1212-121212121212`.

Need a hint? Click here.

Here, you will use an `UPDATE` command. Again, we are updating the row in the `killrvideo.videos` table with the `videoid` of `12121212-1212-1212-1212-121212121212`. The clause you use to remove the tag looks like `tags = tags + { 'Low-rent' }`.\

Want the command? Click here.

```
UPDATE killrvideo.videos SET tags = tags + { 'Low-rent' } WHERE videoid = 12121212-12
```

```
// Write a command to update the row from the videos table
// Execute this cell (click the Run button in the top-right corner)
```

READY

One last time, let's check the contents of the row to see the effects of our command.

READY

Step 9: Execute the following cell.

READY

```
// Execute this cell (click the Run button in the top-right corner)

SELECT * FROM killrvideo.videos WHERE videoid = 12121212-1212-1212-1212-121212121212;
```

Review the results in the previous cell to see that the update worked as expected.

READY

Note (some things to keep in mind about Collections):

To avoid performance problems, only use collections for small-ish numbers of elements
Sets and maps do not incur the read-before-write penalty, but some list operations do. Therefore, when possible, prefer sets to lists
List prepend and append operations are not idempotent, so retrying after a timeout may result in duplicate elements
Collections may only be used in primary keys if they are frozen

READY

Counters

In this section, you will do the following things:

- Investigate the `video_playback_stats` table in `killrvideo`
- Add a row to the `video_playback_stats` table
- Increment the counter of the table to simulate videoing a video

KillrVideo uses the `video_playback_stats` table to keep track of the number of times a video has been viewed. A counter is a great data type for this use-case because counters perform well in Cassandra, and in the rare event where the counter might drop an update, it is not a serious problem for the app or its users.

Let's start by investigating this table.

Step 1: In the following cell, describe the `video_playback_stats` table.

Need a hint? Click here.

You want to use the `DESCRIBE` command to describe only the table. \

Want the command? Click here.

```
DESCRIBE TABLE killrvideo.video_playback_stats;
```

```
// Write a command to allow you to review the table definition for video_playback_stats
// Execute this cell (click the Run button in the top-right corner)
```

READY

READY

Notice that this table has two columns: `videoid` and `views`. Also, notice that `views` is a counter that keeps track of how many times the video has been, uh, viewed.

In this section, we want to update a counter. Just to keep things simple, let's use a contrived `uuid` for the `videoid`. We'll use the value `12121212-1212-1212-1212-121212121212`.

We'll start by verifying that a row for this `videoid` does not yet exist in the table.

Step 2: In the following cell, try to retrieve the row with the `videoid` of `12121212-1212-1212-1212-121212121212`.

5_Advanced_Data_Types

Need a hint? Click here.

For this command, you will use the `SELECT` statement on the `video_playback_stats` table in the `killrvideo` keyspace. You only want the row where the `videoid` is `12121212-1212-1212-1212-121212121212`, but it will be easiest just to grab all the columns.

Want the command? Click here.

```
SELECT * from killrvideo.video_playback_stats WHERE videoid = 12121212-1212-1212-1212
```

```
// Write a command to try to retrieve the row with a videoid of 2121212-1212-1212-1212-121212121
// Execute this cell (click the Run button in the top-right corner)
```

READY

READY

We see “No Data Returned” which allows us to verify that the row with that key is not in the table yet.

Let’s try creating the row with that `videoid`.

Step 3: Create the row by incrementing the `views` counter for the `videoid` `12121212-1212-1212-1212-121212121212`.

Need a hint? Click here.

You will be `UPDATE`ing the row in the `killrvideo.video_playback_stats` with the `videoid` of `12121212-1212-1212-1212-121212121212`. You will increment the counter with code like this: `views = views + 1`.

Want the command? Click here.

```
UPDATE killrvideo.video_playback_stats SET views = views + 1 WHERE videoid = 12121212
```

Thought question: Since we know we cannot `INSERT` a row with a counter, we can only `UPDATE` the row, what will the value of the counter be after we increment it?

```
// Write a command to update the views counter of the row with a videoid of 12121212-1212-1212-1
// Execute this cell (click the Run button in the top-right corner)
```

READY

5_Advanced_Data_Types

Let's see the results of the update!

READY

Step 4: Execute the following cell and inspect the results.

READY

READY

```
// Execute this cell (click the Run button in the top-right corner)
SELECT * FROM killrvideo.video_playback_stats WHERE videoid = 12121212-1212-1212-1212-12121212
```

FINISHED

This time we see the row. We created the row with the `UPDATE` command - an upsert! Notice that the value of the `views` counter is one. That's because, when you create a row by incrementing the counter, it's as if the counter started at zero.

Note (some things to keep in mind about counters):

Counters cannot be part of a primary key
Incrementing or decrementing counters is not idempotent
Incrementing or decrementing a counter is not always guaranteed to work - under high traffic situations, it is possible for one of these operations to get dropped

Took 0 sec. Last updated by anonymous at July 02 2020, 12:53:06 PM. (outdated)

READY

Congratulations!!!!

If you have made it to the end of this notebook successfully, you have additional data types under your belt.

FINISHED

Bonus Challenge: Create a UDT

If you got done early and want something to do while you wait for others, here's a bonus challenge

In this section, you will do the following things:

- Create a UDT to represent video encoding

5_Advanced_Data_Types

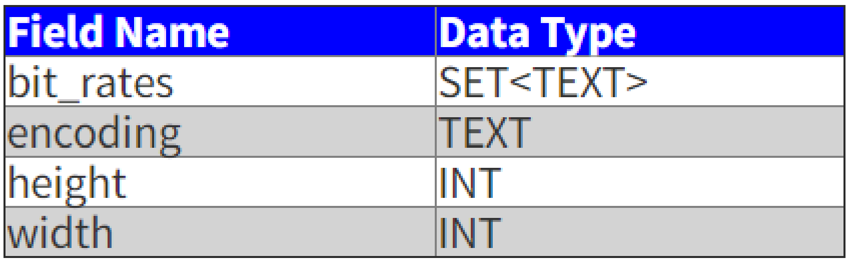
- Alter the KillrVideo `videos` table to use this UDT

- Load some data into the altered `videos` table
- Run a query on the loaded data with the UDT
- Update the value of a row containing a UDT

Here's the pitch:

As KillrVideo grows in popularity, it becomes necessary to support various video formats with different bit rates, encodings and frame sizes. This seems like an ideal use for a UDT. Let's build the UDT and then add it to the `videos` table.

Step 1: In the next cell, write and execute the CQL to create a UDT, named `video_encoding` that contains the fields as described in the following table.



Field Name	Data Type
bit_rates	SET<TEXT>
encoding	TEXT
height	INT
width	INT

Want the solution? Click here.

```
CREATE TYPE IF NOT EXISTS killrvideo.video_encoding (
    bit_rates   SET<TEXT>,
    encoding    TEXT,
    height      INT,
    width       INT
);
```

Took 0 sec. Last updated by anonymous at July 02 2020, 12:53:56 PM. (outdated)

```
// Create the video.encoding UDT in this cell as described.
// Then, to execute this cell, click the Run button in the top-right corner (or SHIFT-ENTER)
```

READY

Step 2: In the next cell, truncate the contents of the `videos` table.

FINISHED

```
TRUNCATE TABLE killrvideo.videos;
```

Took 0 sec. Last updated by anonymous at July 02 2020, 12:55:40 PM. (outdated)

```
// Write the CQL to truncate the videos table
// Then, to execute this cell, click the Run button in the top-rightcorner (or SHIFT-ENTER)
```

READY

```
// Write the CQL to insert row in the videos table
INSERT INTO killrvideo.videos (videoid, tags)
  VALUES(12121212-1212-1212-1212-121212121212, { 'Favorite', 'Fast-paced', 'Funny' });
```

READY

Step 3: In the next cell, alter the `videos` table by adding an `encodings` column of type `FROZEN<video_encoding`.

FINISHED

NOTE: UDTs that contain containers (e.g., sets, lists, etc.) must be declared `FROZEN` to make it explicit that Cassandra serializes the contents of the UDT into a single value for storage.

Want the solution? Click here.

```
ALTER TABLE killrvideo.videos ADD (encoding FROZEN<video_encoding>);
```

Took 2 sec. Last updated by anonymous at July 02 2020, 1:14:29 PM.

```
// Write the CQL to add the encoding column to the videos table
// Then, to execute this cell, click the Run button in the top-right corner (or SHIFT-ENTER)
```

READY

Step: In the following cell, query the videos table for the encoding column value of the row with the videoid of 2644c36e-14bd-11e5-839e-8438355b7e3a.
Inspect the output to see what the UDT looks like.

FINISHED

Want the solution? Click here.

SELECT * FROM killrvideo.videos WHERE videoid = 12121212-1212-1212-1212-121212121212;

Took 0 sec. Last updated by anonymous at July 02 2020, 12:56:52 PM.

```
// Write the CQL command
```

READY

The UDT is `FROZEN` which means if you want to update it, you must replace the entire contents of the UDT. Remember, Cassandra encodes the UDT into something like a single blob.

READY

5_Advanced_Data_Types

Step: In the following cell, write a CQL UPDATE command to add a bit rate of “1500 Kbps” to the `video_encoding` for the row with the `videoid` of `12121212-1212-1212-1212-121212121212`.

- Here is an example of the correct format of the UDT *before* the update of the additional bit rate.

Want the solution? Click here.

```
UPDATE killrvideo.videos
    SET encoding={encoding: '1080p', height: 1080, width: 1920, bit_rates: {'1500 Kbp
    WHERE videoid=12121212-1212-1212-1212-121212121212;
```

```
// Write a CQL update command to add a bit_rate of 1500 Kbps to the row in the videos table with
// Then, to execute this cell, click the Run button in the top-right corner (or SHIFT-ENTER)
```

READY

```
SELECT * FROM killrvideo.videos WHERE videoid = 12121212-1212-1212-1212-121212121212;
```

READY

READY