Exercise 06.03 – Simple Graph Traversal in Studio

In this exercise, you will accomplish the following:

- Use DSE Studio to create a graph
- Investigate the tables DSE uses to store the graph

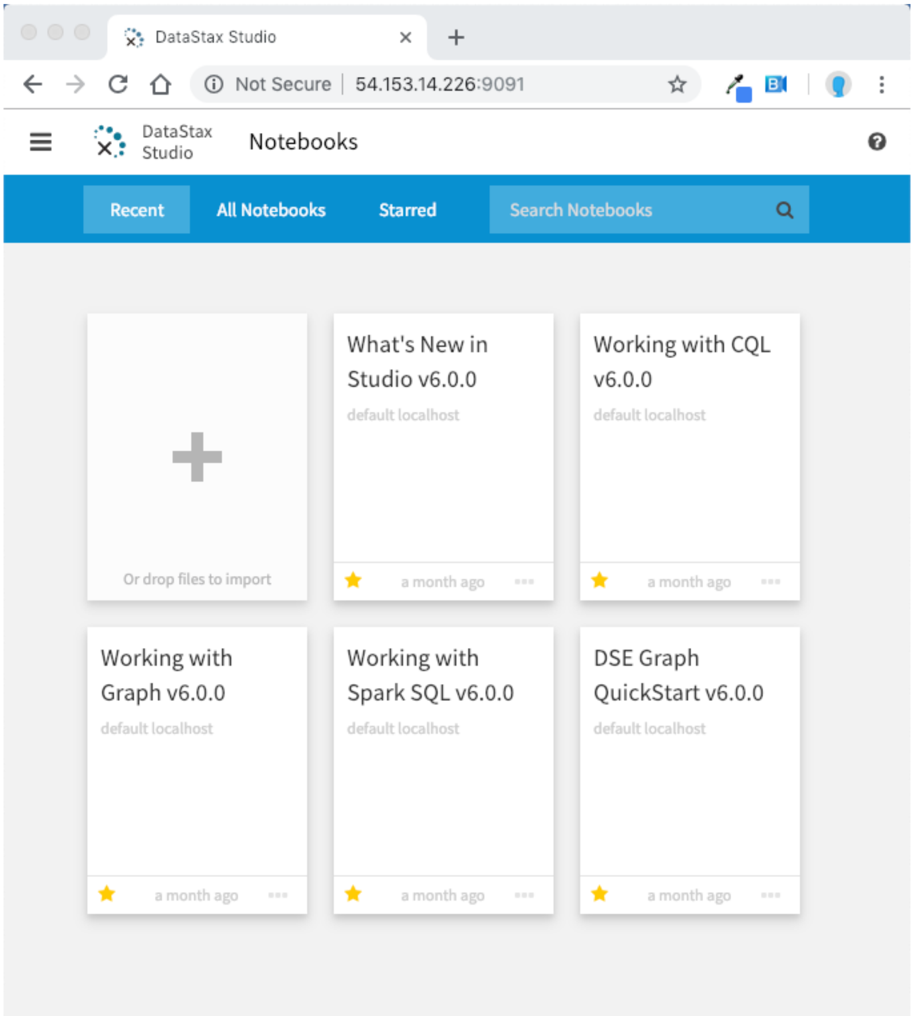
Step 1: Start a DSE Studio Session and Create a Graph

Since DSE Studio is so convenient, use it to create a graph. Later we will also use DSE Studio to investigate the tables that DSE Graph creates.

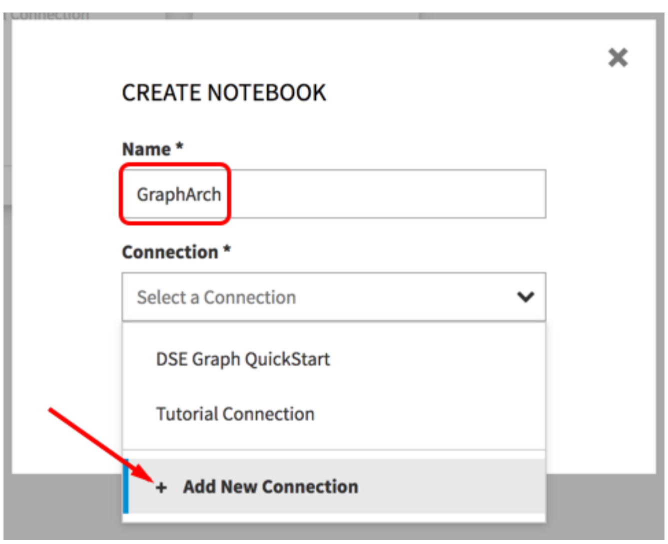
1. In your browser, access the following DSE Studio URL:

```
http://<DSE-node3_IP_address>:9091
```

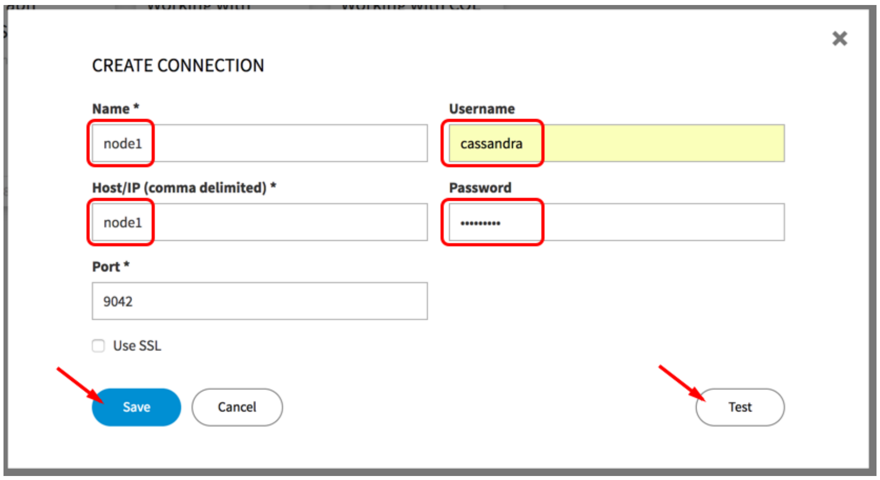
2. When you connect, the browser should look something like the following image. Click on the big plus (+) sign to create a notebook and connect to the cluster:



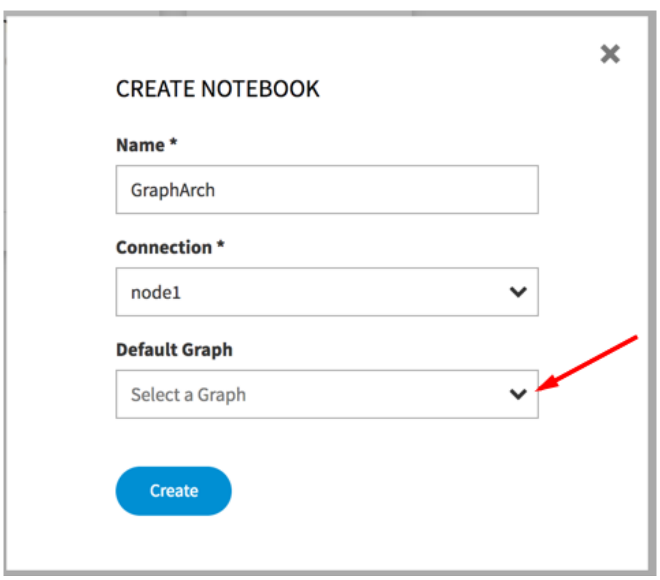
3. In the `CREATE NOTEBOOK` dialog, give your notebook a name. Click on the `Select a Connection` drop down and click on `Add New Connection`.



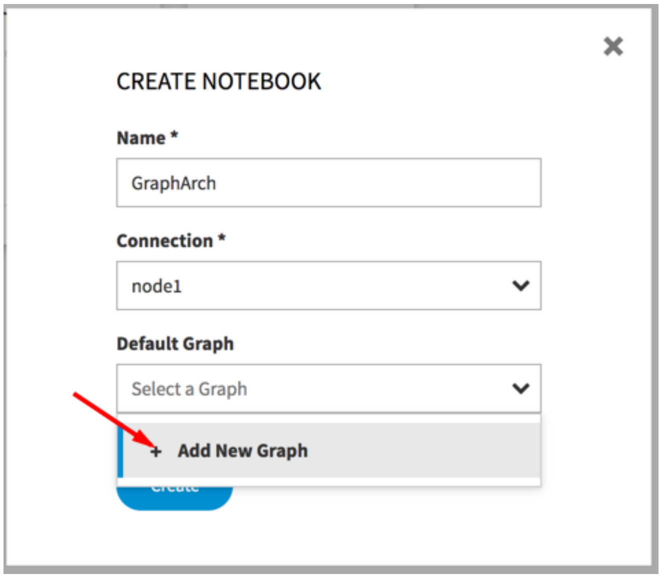
4. In the `CREATE CONNECTION` dialog, fill in a name for the connection, the name of the node in the cluster, and the *cassandra* user name and password (i.e., *cassandra* and *cassandra*). **BE SURE TO SET ALL FOUR VALUES AS NODE3!** The port number can remain as is, which is correct. Once all dialog boxes are filled, click the `Test` button to make sure the connection works. Then, click `Save`.



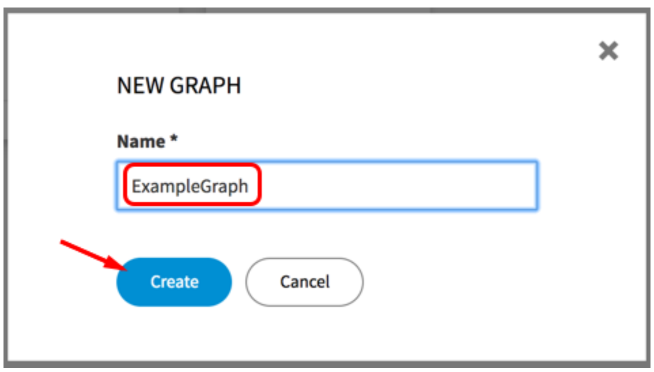
5. Finish filling out the CREATE NOTEBOOK dialog by expanding the Select a Graph drop-down:



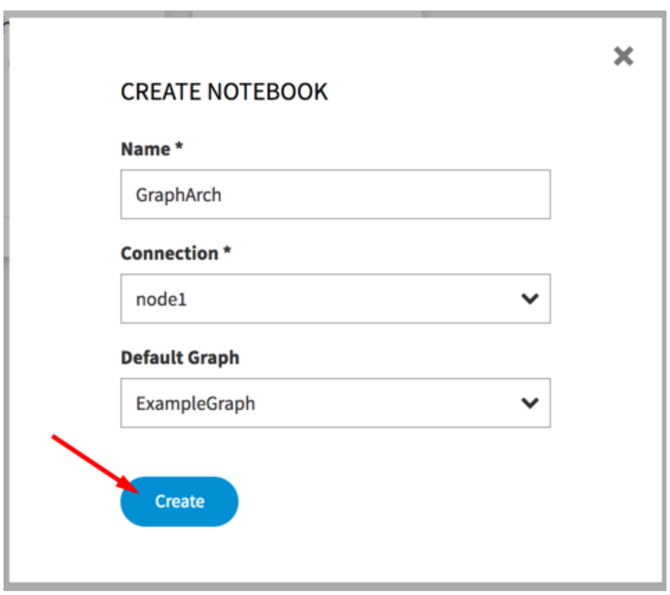
6. Click on `Add a New Graph`:



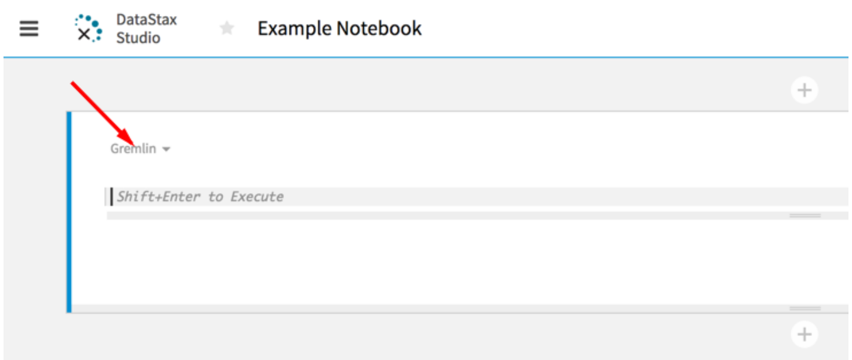
7. Fill in the new graph's name with `ExampleGraph` and click `Create`:



8. Finally, click `Create` to create the notebook:



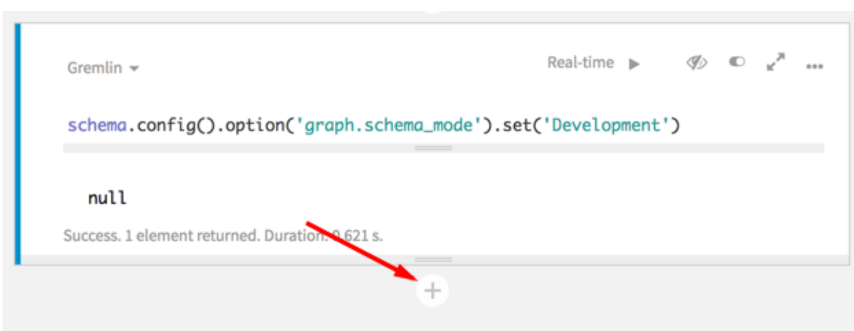
9. You are now in the notebook. Notice that the notebook displays `Gremlin mode`, which is what is used to create an example graph:



10. Create a small example graph in order to explore how DSE stores the graph. Instead of creating the schema up-front, create the schema as the vertices and edges are also created. To do this, set the schema mode to *development*. In the Gremlin cell, enter the following command and execute by clicking on the *Real-time* link in the top right-hand corner:

```
schema.config().option('graph.schema_mode').set('Development')
```

11. Next, create another cell where users enter the graph data. Click on the plus (+) sign:

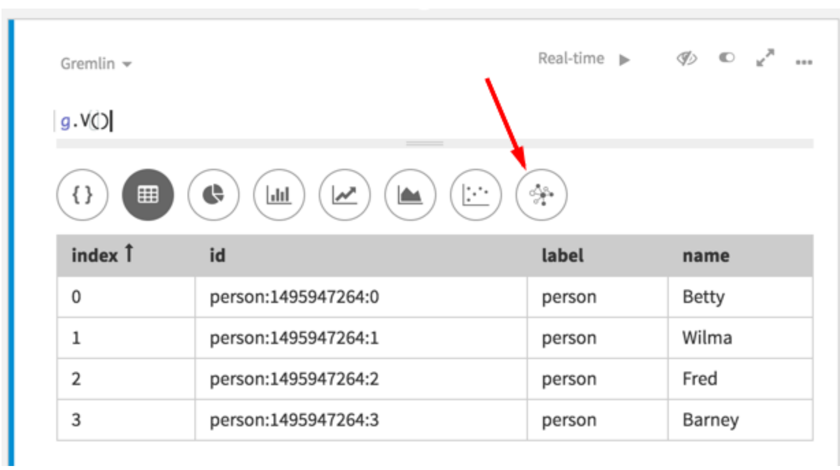


12. In the new cell, enter and execute the following commands and then click on the `Real-time` link:

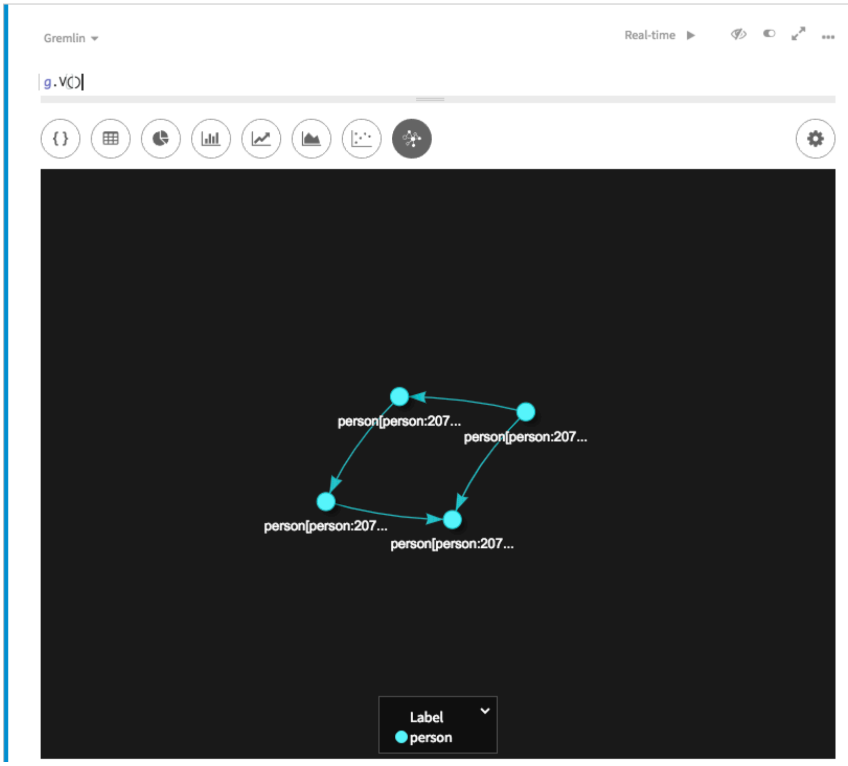
```
fred = graph.addVertex(label, 'person', 'name', 'Fred')
wilma = graph.addVertex(label, 'person', 'name', 'Wilma')
barney = graph.addVertex(label, 'person', 'name', 'Barney')
betty = graph.addVertex(label, 'person', 'name', 'Betty')
```

```
fred.addEdge('friends_with', barney)
fred.addEdge('married_to', wilma)
barney.addEdge('married_to', betty)
betty.addEdge('friends_with', wilma)
```

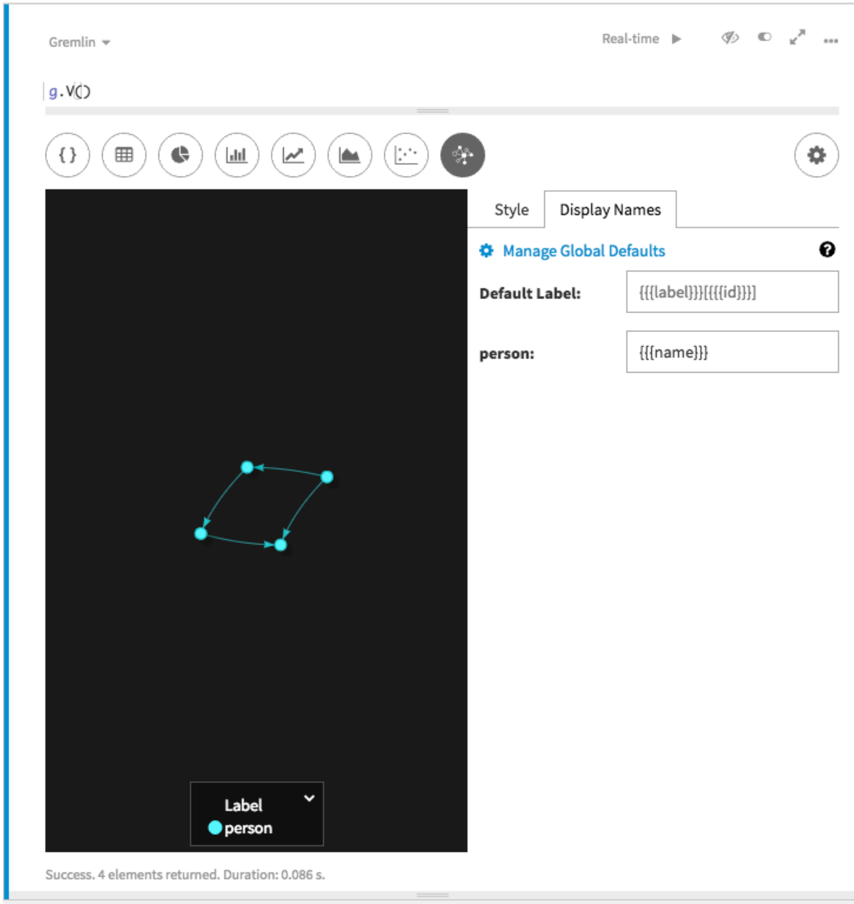
13. Create a new Gremlin cell and enter and execute the command: `g.V()`. Note: V is a capital V as in "Victor". Then, click on the `Graph` icon:



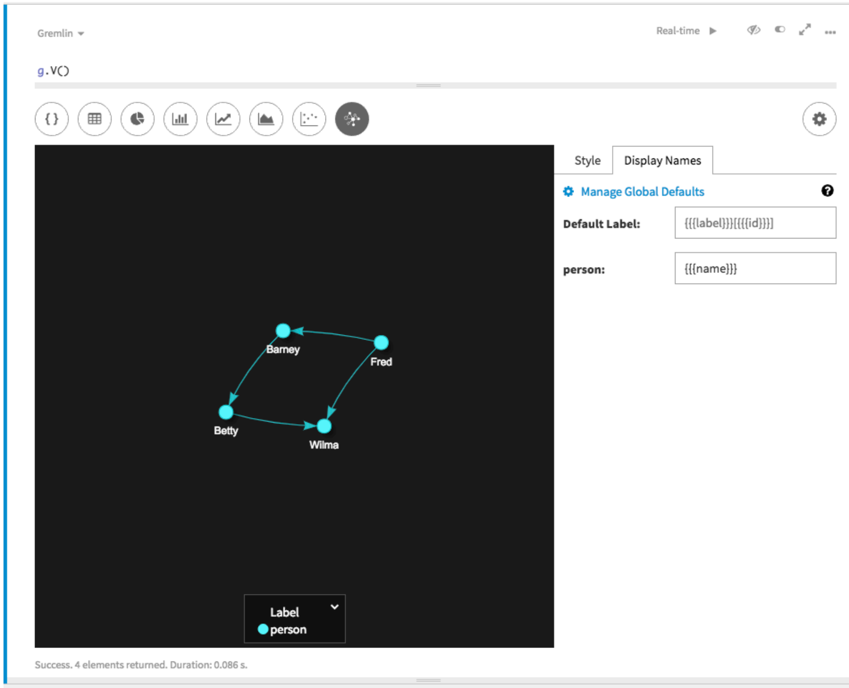
14. To view the actual graph, change the labels by selecting the `Graph Settings` menu:



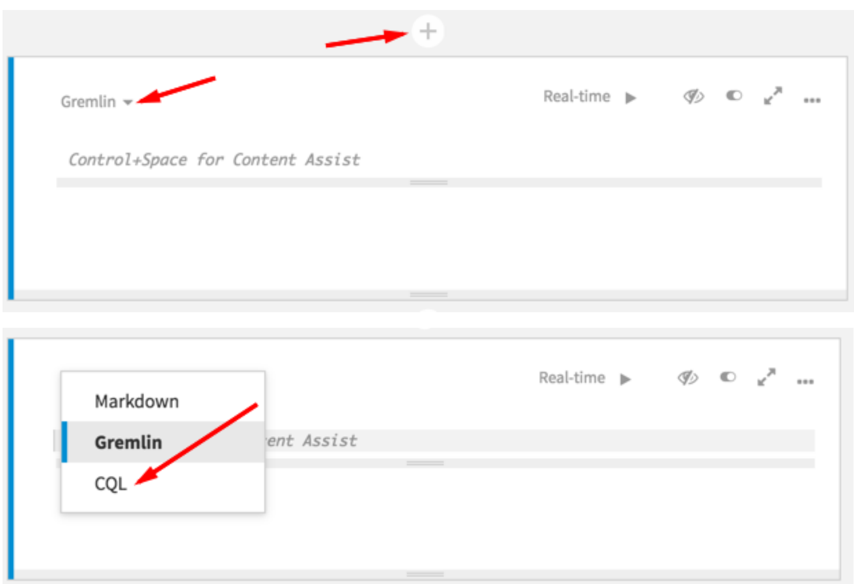
15. Click on the `Display Names` tab. Fill in the field labeled `person:` with `{{{name}}}`. Then, click the `Graph Settings` menu icon to close the menu:



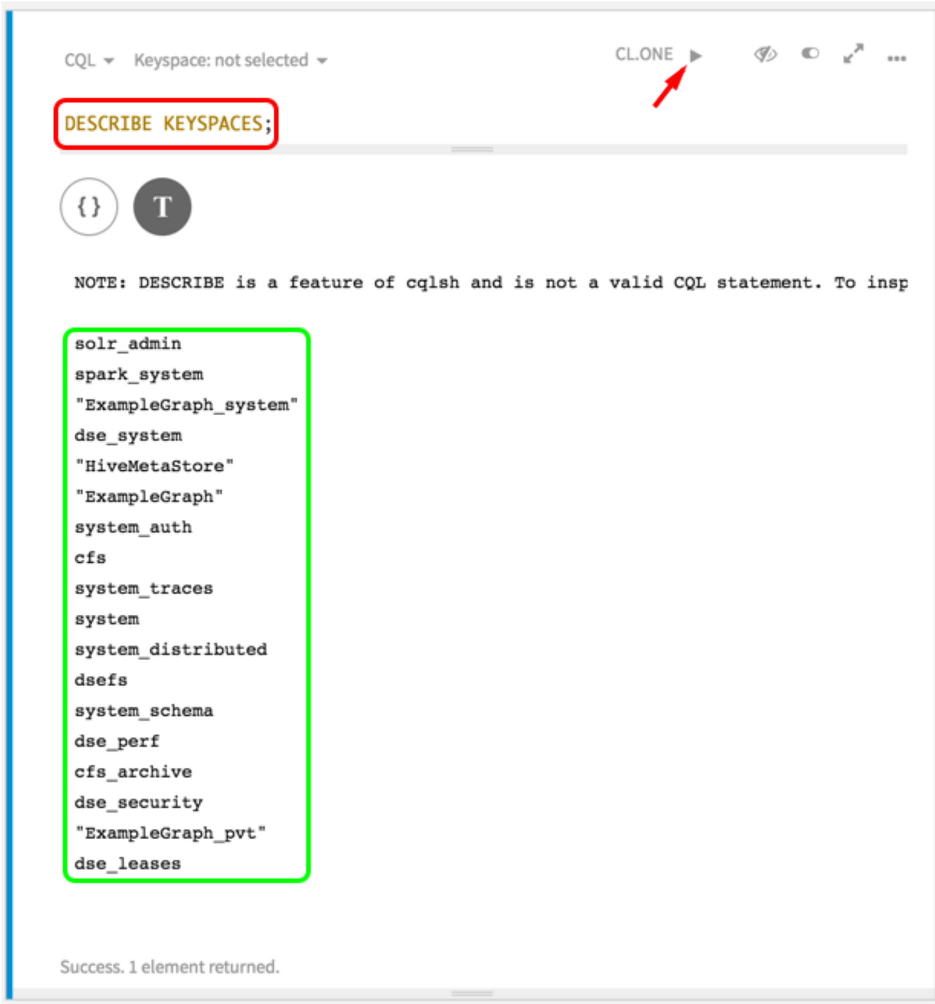
16. There are now four vertices and associated edges created in the previous step. Hover over each edge to view its label. Increase the browser screen size to view additional information.



17. Investigate how DSE stores the graph. Create another cell in your notebook but make this one a `CQL` cell. Click on the plus (+) sign to create the cell and click on the `Select Language` menu to select `CQL`:



18. Look at the available keyspaces. In this `CQL` cell, enter and execute `DESCRIBE KEYSPACES;` by clicking on the `Clone` button.



The list of keyspaces shows the usual system-type keyspaces. Additionally, there are three keyspaces that start with the name of your graph. In this example, the keyspaces are "ExampleGraph", "ExampleGraph_system", and "ExampleGraph_pvt". Note that the quotes are part of the keyspace name. The quotes make it so that the keyspace names are case-sensitive.

The "ExampleGraph" keyspace contains the actual data (i.e., properties and edges) for your graph. The "ExampleGraph_system" keyspace contains schema data for the graph. The "ExampleGraph_pvt" keyspace would normally contain information about super-nodes, but as there are presently no super-nodes in this graph, this keyspace is either empty or not present.

There is other graph-related information scattered within the tables of the *system* keyspace. This is important to remember when backing up all graph keyspaces.

19. Start investigating the "ExampleGraph" keyspace. Create a CQL cell in your notebook and execute the following:

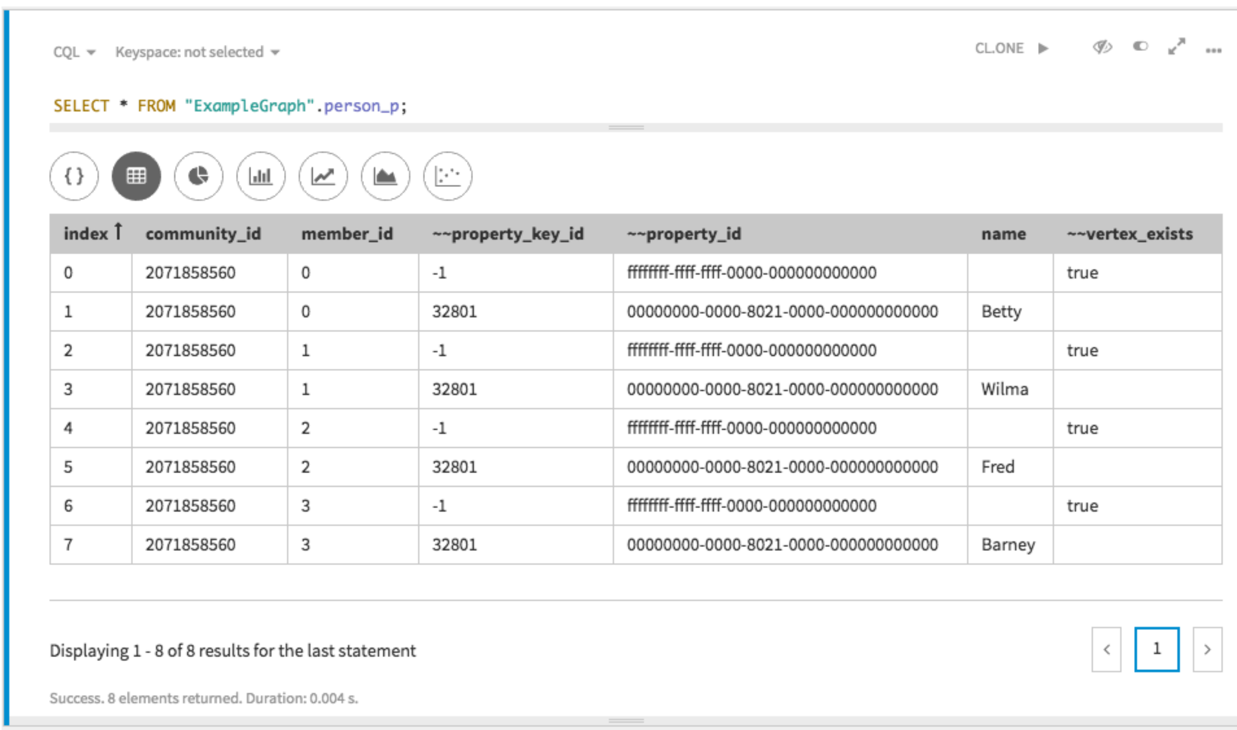
```
DESCRIBE KEYSPACE "ExampleGraph";
```

Expand the browser window, if necessary, to view the CQL cell as it does not wrap lines by default.

Note there are three tables in this keyspace: `id_allocation`, `person_e`, and `person_p`. The `id_allocation` table contains information DSE uses to allocate standard vertex IDs. Since users typically use custom IDs to improve locality, ignore this table for now.

20. Examine the `person_p` table. Create another CQL cell and execute the following command. Notice the keyspace must be included as part of the query. You will be unable to select the graph-related keyspaces from the keyspaces dropdown as Studio intentionally hides these. This is to prevent users from misconfiguring the data much as we are doing now.

```
SELECT * FROM "ExampleGraph".person_p;
```



index	community_id	member_id	~~property_key_id	~~property_id	name	~~vertex_exists
0	2071858560	0	-1	ffffffff-ffff-ffff-0000-000000000000		true
1	2071858560	0	32801	00000000-0000-8021-0000-000000000000	Betty	
2	2071858560	1	-1	ffffffff-ffff-ffff-0000-000000000000		true
3	2071858560	1	32801	00000000-0000-8021-0000-000000000000	Wilma	
4	2071858560	2	-1	ffffffff-ffff-ffff-0000-000000000000		true
5	2071858560	2	32801	00000000-0000-8021-0000-000000000000	Fred	
6	2071858560	3	-1	ffffffff-ffff-ffff-0000-000000000000		true
7	2071858560	3	32801	00000000-0000-8021-0000-000000000000	Barney	

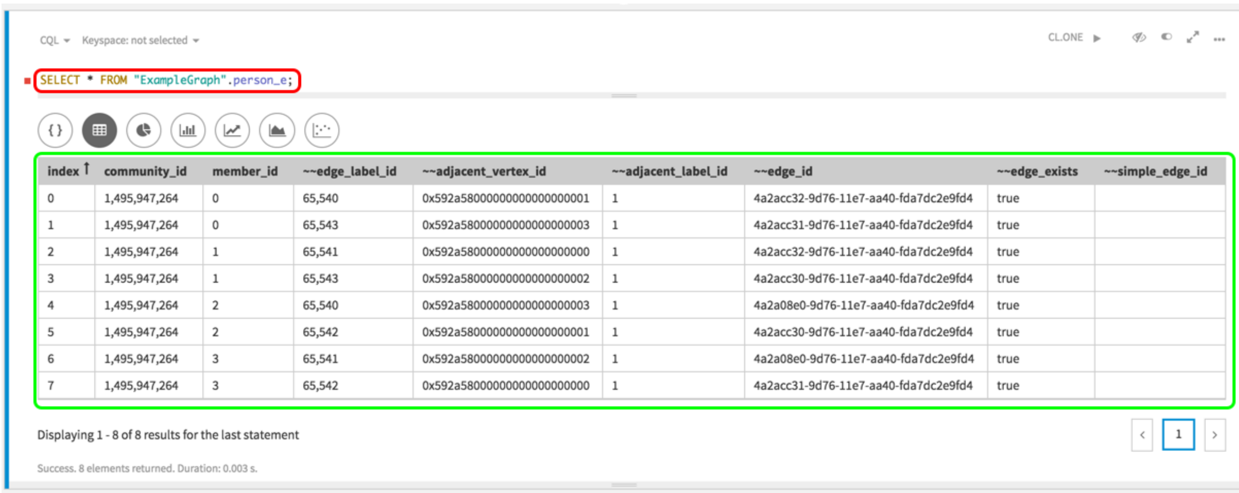
These are the properties associated with the example graph. One of the property types indicates the vertex exists as is shown in the final column. The other properties are the names associated with the vertices. The `community_id` and the `member_id` make up the vertex ID.

Refer back to the table description and you will note that the `community_id` is the partition key and the `member_id` is a clustering column. The `~~property_key_id` and the `~~property_id` are also clustering columns that act a bit like foreign keys in the "ExampleGraph_system" keyspace and help to identify the property type.

You may be wondering why, in the example shown, Betty gets the member ID of 0 instead of Fred, since it appears that we created Fred first. This is because Studio creates all vertices in a group concurrently. So, the order of the creation of each vertex is not necessarily sequential. You may see a different order than that illustrated in this example.

21. Now, examine the *person_e* table. Create another CQL cell and execute the following query:

```
SELECT * FROM "ExampleGraph".person_e;
```



index	community_id	member_id	~~edge_label_id	~~adjacent_vertex_id	~~adjacent_label_id	~~edge_id	~~edge_exists	~~simple_edge_id
0	1,495,947,264	0	65,540	0x592a5800000000000000001	1	4a2acc32-9d76-11e7-aa40-fda7dc2e9fd4	true	
1	1,495,947,264	0	65,543	0x592a5800000000000000003	1	4a2acc31-9d76-11e7-aa40-fda7dc2e9fd4	true	
2	1,495,947,264	1	65,541	0x592a5800000000000000000	1	4a2acc32-9d76-11e7-aa40-fda7dc2e9fd4	true	
3	1,495,947,264	1	65,543	0x592a5800000000000000002	1	4a2acc30-9d76-11e7-aa40-fda7dc2e9fd4	true	
4	1,495,947,264	2	65,540	0x592a5800000000000000003	1	4a2a08e0-9d76-11e7-aa40-fda7dc2e9fd4	true	
5	1,495,947,264	2	65,542	0x592a5800000000000000001	1	4a2acc30-9d76-11e7-aa40-fda7dc2e9fd4	true	
6	1,495,947,264	3	65,541	0x592a5800000000000000002	1	4a2a08e0-9d76-11e7-aa40-fda7dc2e9fd4	true	
7	1,495,947,264	3	65,542	0x592a5800000000000000000	1	4a2acc31-9d76-11e7-aa40-fda7dc2e9fd4	true	

Displaying 1 - 8 of 8 results for the last statement

Success. 8 elements returned. Duration: 0.003 s.

This table displays the edges in the graph. Notice there are two entries for each edge created. The two entries are necessary to represent the edge adjacent to each vertex. Also, note the `community_id` and `member_id` columns. The values in these columns correlate to the `community_id` and `member_id` columns of the properties table. Since Betty has `member_id` 0, the first two rows of the edge table show the edges adjacent to Betty (i.e., that she knows Wilma and is married to Barney).

The `adjacent_vertex_id` column indicates to whom the other end of the edge is attached. These values show up as hexadecimal values, but if we examine them more closely and convert these values to decimal, we see that they correlate to the `community_id` and `member_id` of the connecting vertex. So, for example, row one shows an edge that connects Betty (`member_id 0`) to Wilma (`member_id 1`).

The edge_label_id column indicates the type of relationship, or in other words, the edge label. The actual name of this label is in the shared_data table within the "ExampleGraph_system" keyspace. The edge_label_id values use the low-order bit to indicate the direction of the relationship. In the example, 65,540 represents the "friends with" relationship, while 65,541 represents the inverse relationship of "is befriended by".

Those are the fundamentals of the DSE Graph architecture. As mentioned, in addition to the three graph-specific keyspaces, some additional graph-related data is stored in the system keyspace. Because of these additional complexities, users should rest the urge and never modify or drop graph keyspaces or tables directly using CQL. Instead, use Gremlin via DSE Studio, Gremlin Console, or Graph drivers (including Spark DSEGraphFrames) to modify graph schema and data.

END OF EXERCISE