KATANA GRAPH

- ☐ Andrew Tan
- ☐ James Coffey
- ☐ Kamesh Peri

Build a Graph from Tables in Databricks

This quick start guide will show you how to build a graph in the Katana Graph Intelligence Platform (KGIP) from source tables in Databricks.

Prepare the data

A graph is made of nodes and edges. Your Databricks tables can describe various types of nodes and edges. The records in these tables should represent one node or edge. For example, you could have a table for Customers and another for Products. A third table could describe Orders which connect customers to products.

Set up the connection

We assume your Databricks cluster and KGIP cluster are already set up. To make the data accessible outside of Databricks, you must use Delta Sharing. The Databricks Delta Sharing documentation guides you through the process. The key goals are:

- Expose the source tables in a share.
- Get a credential file for that share. We will call this file `profile.json`.

You can confirm that the connection works by spot-checking a table from a notebook in the KGIP cluster:

```
import dask_deltasharing

customers = dask_deltasharing.load_as_dask(
    "profile.json#my-share.default.customers"
)
customers.head()
```

Creating a graph

To build a graph from the source data, you will have to specify what kind of data is in each table and how it's supposed to be put together. This specification is done through the KGIP DataFrameImporter interface.

So, in our example, the code to build an RDG would look like this:

```
# Grab the data through Delta Sharing.
import dask_deltasharing

customers = dask_deltasharing.load_as_dask(
    "profile.json#my-share.default.customers"
)
products = dask_deltasharing.load_as_dask(
    "profile.json#my-share.default.products"
)
orders =
dask_deltasharing.load_as_dask("profile.json#my-share.defaul
t.orders")

# Build a graph. Same as with any other data source.
```

```
from katana import remote
from katana.remote import import_data

client = remote.Client()
graph = client.create_graph()
with import_data.DataFrameImporter(graph) as df_importer:
    df_importer.nodes_dataframe(
        customers, id_column="name", id_space="Customer"
    )
    df_importer.nodes_dataframe(products, id_column="id",
id_space="Product")
    df_importer.edges_dataframe(
        orders,
        source_id_space="Customer",
        source_column="customer",
        destination_id_space="Product",
        destination_column="product_id",
    )
    df_importer.insert()
```

Now your graph is ready for anything! Run a few queries to confirm it is as expected.

```
graph.query("MATCH (v) RETURN COUNT(v)")
graph.query("MATCH (v) RETURN v LIMIT 1")
graph.query("MATCH ()-[r]->() RETURN COUNT(r)")
graph.query("MATCH ()-[r]->() RETURN r LIMIT 1")
```