Machine Learning Workshop with Oracle Graph

Contents

Explore how to build a Bank Fraud Detection engine using Oracle Graph on a real-world dataset residing in Oracle Autonomous Database.

About This Workshop

This workshop walks you through the steps to build a Bank Fraud Detection engine using Oracle Graph on a real-world dataset residing in Oracle Autonomous Database. You will access to a bank dataset, use Graph Server, create a Property Graph data model, perform graph visualization and graph analysis.

Workshop prerequisites:

- Familiarity with Database is desirable, but not required
- Some understanding of cloud and database terms is helpful
- Familiarity with Oracle Cloud Infrastructure (OCI) is helpful
- Familiarity with Graph technologies is helpful
- Internet connection to use a web browser.

Workshop details:

Together with Oracle experts you will be able to learn and experience the advantages of Oracle's converged database for transactional uses with advanced graph analytics, which will help you innovate, improving the management, efficiency and performance of your data models.

In this workshop, we will teach you to develop graph analytics (Property Graph) and you will learn about the tools to transform a relational model into a graph. You will explore the use of the PGQL query language, the included advanced analytics algorithms, and the GraphViz visualization tool.We will also teach you to connect opensource graph tool like Vis.js or D3.js to our property graph data model.

Scenario Explanation

While no fraud prevention measures can ever be perfect, significant opportunity for improvement lies in looking beyond the individual data points, to the connections that link them. Oftentimes these connections go unnoticed until it is too late— something that is unfortunate, as these connections oftentimes hold the best clues.

While the exact details behind each first-party fraud collusion vary from operation to operation, the pattern below illustrates how fraud rings commonly operate:

- A group of two or more people organize into a fraud ring
- The ring shares a subset of legitimate contact information, for example phone numbers and addresses, combining them to create a number of synthetic identities
- Ring members open accounts using these synthetic identities
- New accounts are added to the original ones: unsecured credit lines, credit cards, overdraft protection, personal loans, etc.
- The accounts are used as normally, with regular purchases and timely payments
- Banks increase the revolving credit lines over time, due to the observed responsible credit behavior
- One day the ring “busts out”, coordinating their activity, maxing out all of their credit lines, and disappearing
- Sometimes fraudsters will go a step further and bring all of their balances to zero using fake checks immediately before the prior step, doubling the damage
- Collections processes ensue, but agents are never able to reach the fraudster
- The uncollectible debt is written off

Scenario Solution

Oracle Graph databases offer new methods of uncovering fraud rings and other sophisticated scams with a high-level of accuracy, and are capable of stopping advanced fraud scenarios in real-time.

We will use the Oracle Graph Database solution in order to resolve this problem.

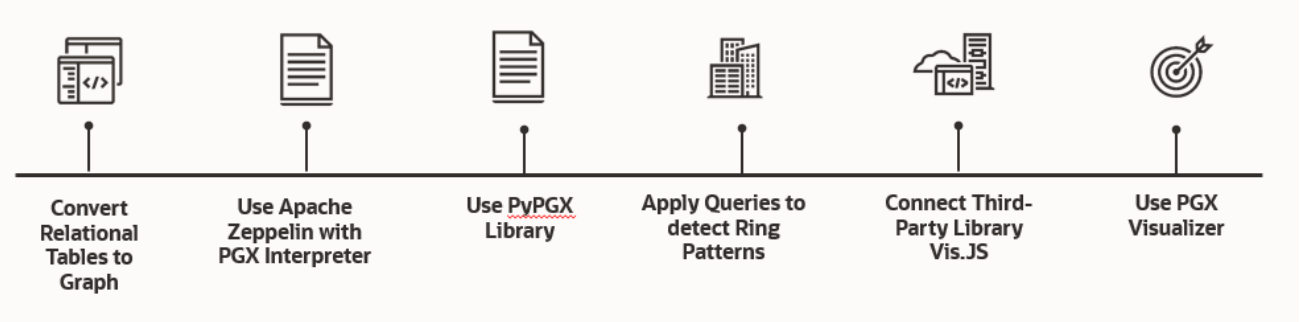
Augmenting one's existing fraud detection infrastructure to support ring detection can be done by running appropriate entity link analysis queries using a graph database, and running checks during key stages in the customer & account lifecycle, such as:

- At the time the account is created
- During an investigation
- As soon as a credit balance threshold is hit
- When a check is bounced

Real-time graph traversals tied to the right kinds of events can help banks identify probable fraud rings: during or even before the Bust-Out occurs.

Workshop Journey Map

To demonstrate the Oracle Graph functionalities, the workshop has been divided in different steps.



The documentation for this workshop has been published in the next github account:

https://github.com/operard/mlgraph

You can find the Documentation of Oracle Graph in the next URL:

https://docs.oracle.com/cd/E56133_01/latest/prog-guides/index.html

The documentation for PGQL (Property Graph Query Language) is available in the next URLs:

https://pgql-lang.org/

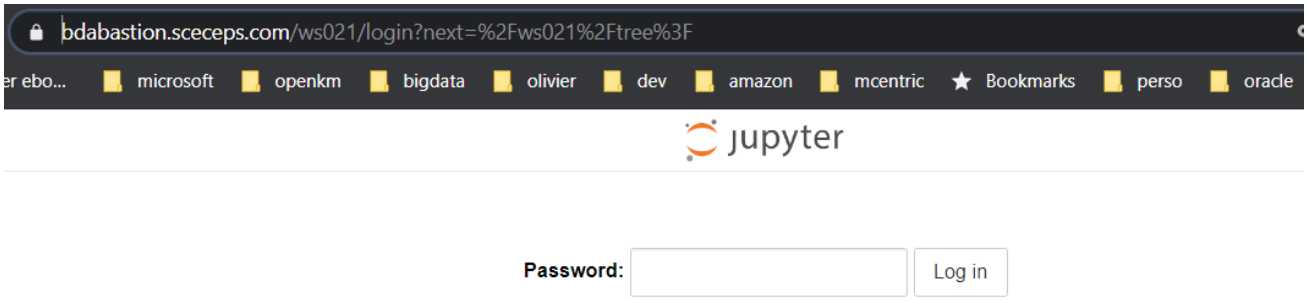
https://github.com/oracle/pgql-lang

Each developer will use an account to access to the Oracle Graph Lab:

User: **workshopxxx** (xxx between 001 to 100)
Pwd: **welcome1**

The url to access to the Oracle Graph Lab is:

https://bdabastion.sceceps.com/wsxxx/



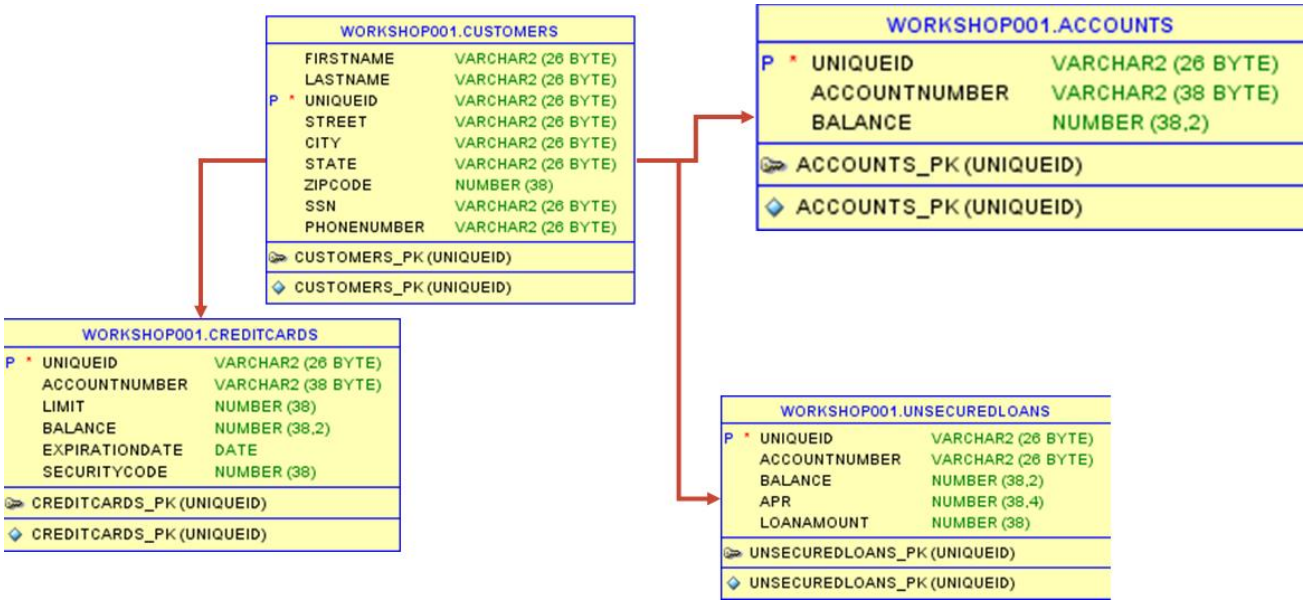
The password to access to the Lab environment is: "**welcome1**"

Step 1: Convert Relational Data Model to Graph Data Model.

The bank stores the Customers data in a relational Data Model. The Banking or financial Data model stores the next information:

- Customers
- Accounts
- Credit Cards
- Unsecured Loans

The relational 3FN data model below represents how the data actually looks to the oracle database:




```
CREATE PROPERTY GRAPH bank
 VERTEX TABLES (
   ACCOUNTS KEY(UNIQUEID) PROPERTIES ALL COLUMNS,
   CUSTOMERS as CUST KEY(UNIQUEID) PROPERTIES
(FIRSTNAME,LASTNAME),
   ADDRESS KEY(STREET,CITY,STATE,ZIPCODE) PROPERTIES
(STREET,CITY,STATE,ZIPCODE),
   PHONENUMBERS KEY(PHONENUMBER) PROPERTIES (PHONENUMBER),
   SSN KEY(SSN) PROPERTIES ALL COLUMNS,
   CREDITCARDS KEY(UNIQUEID) PROPERTIES ALL COLUMNS,
   UNSECUREDLOANS KEY(UNIQUEID) PROPERTIES ALL COLUMNS
 )
 EDGE TABLES (
  ACCOUNTS SOURCE KEY(UNIQUEID) REFERENCES CUST  DESTINATION
KEY(UNIQUEID) REFERENCES ACCOUNTS LABEL HAS_BANKACCOUNT,
  CUST_ADDRESS SOURCE KEY(UNIQUEID) REFERENCES CUST DESTINATION
KEY(STREET,CITY,STATE,ZIPCODE) REFERENCES ADDRESS    LABEL
HAS_ADDRESS,
  CUST_PHONENUMBERS SOURCE KEY(UNIQUEID) REFERENCES CUST
DESTINATION KEY(PHONENUMBER) REFERENCES PHONENUMBERS
```

```
   LABEL HAS_PHONENUMBER,
  CUST_SSN SOURCE KEY(UNIQUEID) REFERENCES CUST DESTINATION
KEY(SSN) REFERENCES SSN
   LABEL HAS_SSN,
  CREDITCARDS SOURCE KEY(UNIQUEID) REFERENCES CUST DESTINATION
KEY(UNIQUEID) REFERENCES CREDITCARDS
   LABEL HAS_CREDITCARDS,
  UNSECUREDLOANS SOURCE KEY(UNIQUEID) REFERENCES CUST
DESTINATION KEY(UNIQUEID) REFERENCES UNSECUREDLOANS
   LABEL HAS_UNSECUREDLOANS
 );
```

In order to create the Graph Data Model (Property Graph Data Model), we will use the **SQLCL** Tool of Oracle Database.

To execute this tool, we will open the Shell Terminal from jupyter tool:

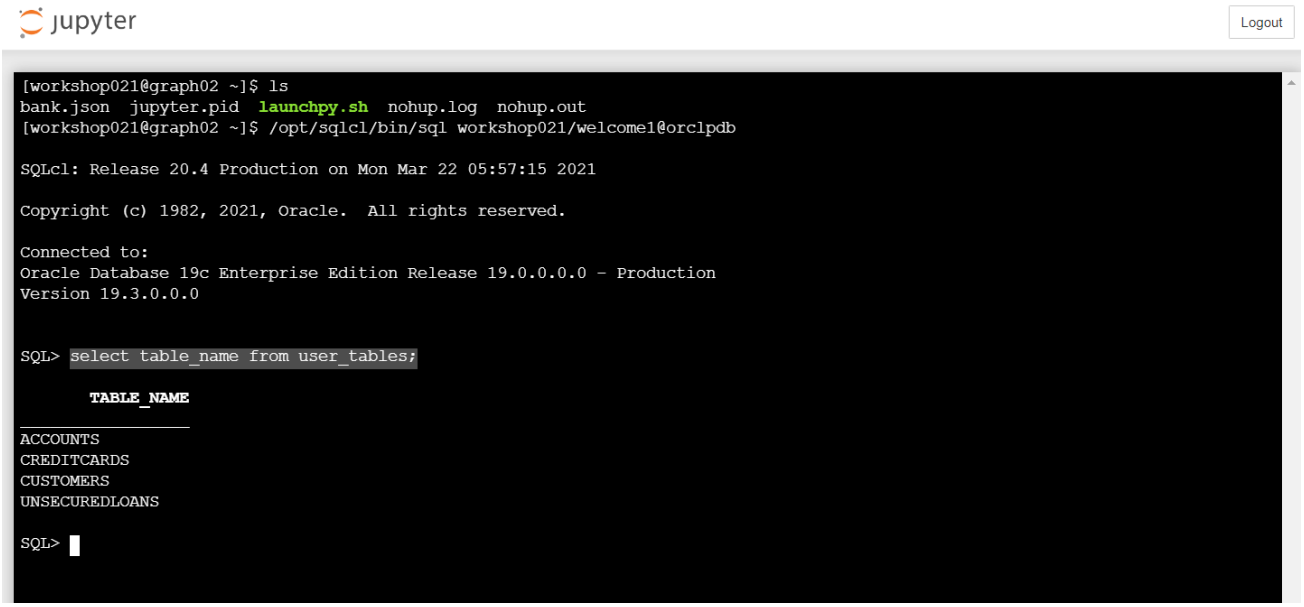
And Jupyter opens the next terminal:

Now, we can execute the next command:

```
/opt/sqlcl/bin/sql workshop**xxx**/welcome1@orclpdb
```

Check tables in your schema:

```
select table_name from user_tables;
```



Now, we will activate the PGQL language inside the SQL:

```
pgql auto on
```

Create the property graph schema from Relational Data Model:

```
CREATE PROPERTY GRAPH bank  VERTEX TABLES (    ACCOUNTS
KEY(UNIQUEID) PROPERTIES ALL COLUMNS,    CUSTOMERS as CUST
KEY(UNIQUEID) PROPERTIES (FIRSTNAME,LASTNAME),    ADDRESS
KEY(STREET,CITY,STATE,ZIPCODE) PROPERTIES
(STREET,CITY,STATE,ZIPCODE),    PHONENUMBERS KEY(PHONENUMBER)
PROPERTIES (PHONENUMBER),    SSN KEY(SSN) PROPERTIES ALL
COLUMNS,    CREDITCARDS KEY(UNIQUEID) PROPERTIES ALL
COLUMNS,    UNSECUREDLOANS KEY(UNIQUEID) PROPERTIES ALL
COLUMNS  )  EDGE TABLES (   ACCOUNTS     SOURCE KEY(UNIQUEID)
REFERENCES CUST     DESTINATION KEY(UNIQUEID) REFERENCES
ACCOUNTS     LABEL HAS_BANKACCOUNT,   CUST_ADDRESS    SOURCE
KEY(UNIQUEID) REFERENCES CUST     DESTINATION
KEY(STREET,CITY,STATE,ZIPCODE) REFERENCES ADDRESS     LABEL
HAS_ADDRESS,   CUST_PHONENUMBERS    SOURCE KEY(UNIQUEID)
REFERENCES CUST     DESTINATION KEY(PHONENUMBER) REFERENCES
PHONENUMBERS     LABEL HAS  PHONENUMBER,   CUST  SSN    SOURCE
KEY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(SSN) REFERENCES
SSN     LABEL HAS_SSN,   CREDITCARDS    SOURCE KEY(UNIQUEID)
REFERENCES CUST     DESTINATION KEY(UNIQUEID) REFERENCES
CREDITCARDS     LABEL HAS_CREDITCARDS,   UNSECUREDLOANS    SOURCE
KEY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(UNIQUEID)
REFERENCES UNSECUREDLOANS     LABEL HAS_UNSECUREDLOANS  );
```

```
SQL> pgql auto on

PGQL Auto enabled for graph=[null], execute=[true], translate=[false]
PGQL> CREATE PROPERTY GRAPH bank  VERTEX TABLES (    ACCOUNTS KEY(UNIQUEID) PROPERTIES ALL COLUMNS,    CUSTOMERS as CUST KEY(U
NIQUEID) PROPERTIES (FIRSTNAME,LASTNAME),    ADDRESS KEY(STREET,CITY,STATE,ZIPCODE) PROPERTIES (STREET,CITY,STATE,ZIPCODE),
 PHONENUMBERS KEY(PHONENUMBER) PROPERTIES (PHONENUMBER),    SSN KEY(SSN) PROPERTIES ALL COLUMNS,    CREDITCARDS KEY(UNIQUEID)
PROPERTIES ALL COLUMNS,    UNSECUREDLOANS KEY(UNIQUEID) PROPERTIES ALL COLUMNS  )  EDGE TABLES (   ACCOUNTS     SOURCE KEY(UNI
QUEID) REFERENCES CUST     DESTINATION KEY(UNIQUEID) REFERENCES ACCOUNTS     LABEL HAS_BANKACCOUNT,   CUST_ADDRESS    SOURCE K
EY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(STREET,CITY,STATE,ZIPCODE) REFERENCES ADDRESS     LABEL HAS_ADDRESS,   CUST_P
HONENUMBERS    SOURCE KEY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(PHONENUMBER) REFERENCES PHONENUMBERS     LABEL HAS_PHO
NENUMBER,   CUST_SSN    SOURCE KEY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(SSN) REFERENCES SSN     LABEL HAS_SSN,   CRED
ITCARDS    SOURCE KEY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(UNIQUEID) REFERENCES CREDITCARDS     LABEL HAS_CREDITCARDS
,   UNSECUREDLOANS    SOURCE KEY(UNIQUEID) REFERENCES CUST     DESTINATION KEY(UNIQUEID) REFERENCES UNSECUREDLOANS     LABEL H
AS_UNSECUREDLOANS  );

Graph created

PGQL>
```

Now you can go out PGQL in order to check the Property Graph Data Model:

```
pgql auto off
```

Check tables in your schema:

```
select table_name from user_tables;
```

```
PGQL> pgql auto off

PGQL Auto disabled
SQL> select table_name from user_tables;

       TABLE_NAME
__________________
ACCOUNTS
BANKGE$
BANKGT$
BANKIT$
BANKSS$
BANKVD$
BANKVT$
CREDITCARDS
CUSTOMERS
UNSECUREDLOANS

10 rows selected.
```

You can see that the “bank” property graph Data Model has created new tables in your database Schema:

- BANKGE$
- BANKGT$
- BANKIT$
- BANKSS$
- BANKVD$
- BANKVT$

Now, we will activate the PGQL language inside the SQL in order to delete the Graph Data Model created “BANK”:

```
pgql auto on
```

You can delete the Graph Data Model using the next commands (change the XXX by your workshop user ID):

```
/opt/sqlcl/bin/sql workshopXXX/welcome1@orclpdb
PGQL AUTO ON
DROP PROPERTY GRAPH BANK;
```

You can create your property graph Data Model „BANK“ using the next command line:

```
/opt/sqlcl/bin/sql workshopXXX/welcome1@orclpdb @pgbank_create.sql
```

Step 2: Use Apache Zeppelin to access Graph Data Model.

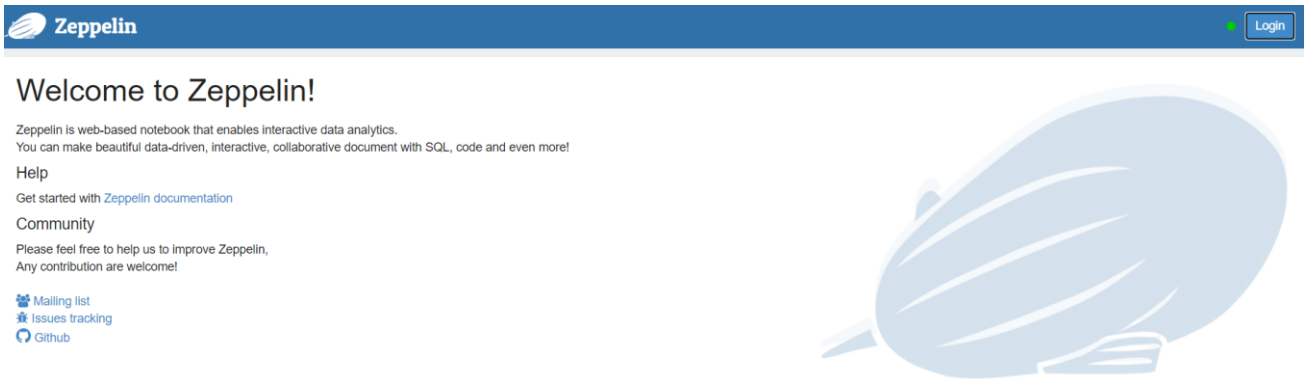
Apache Zeppelin is an open source framework to develop notebooks using different programming languages using different interpreters available in the next URL:

http://zeppelin.apache.org/

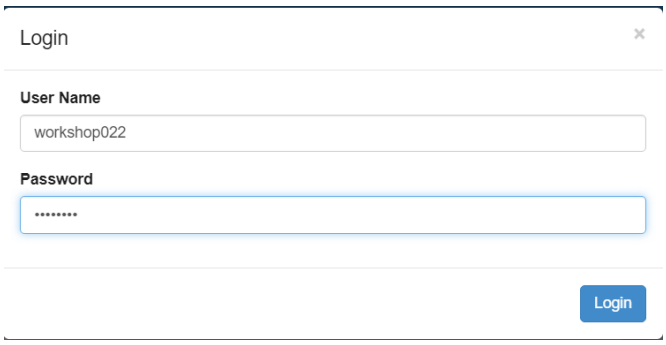
All users "workshopXXX" have been distributed in different Labs:

Users	Zeppelin URL
Workshop001 to workshop020	https://bdabastion.sceceps.com/zp1/
Workshop021 to workshop040	https://bdabastion.sceceps.com/zp2/
Workshop041 to workshop060	https://bdabastion.sceceps.com/zp3/
Workshop061 to workshop080	https://bdabastion.sceceps.com/zp4/
Workshop081 to workshop100	https://bdabastion.sceceps.com/zp5/

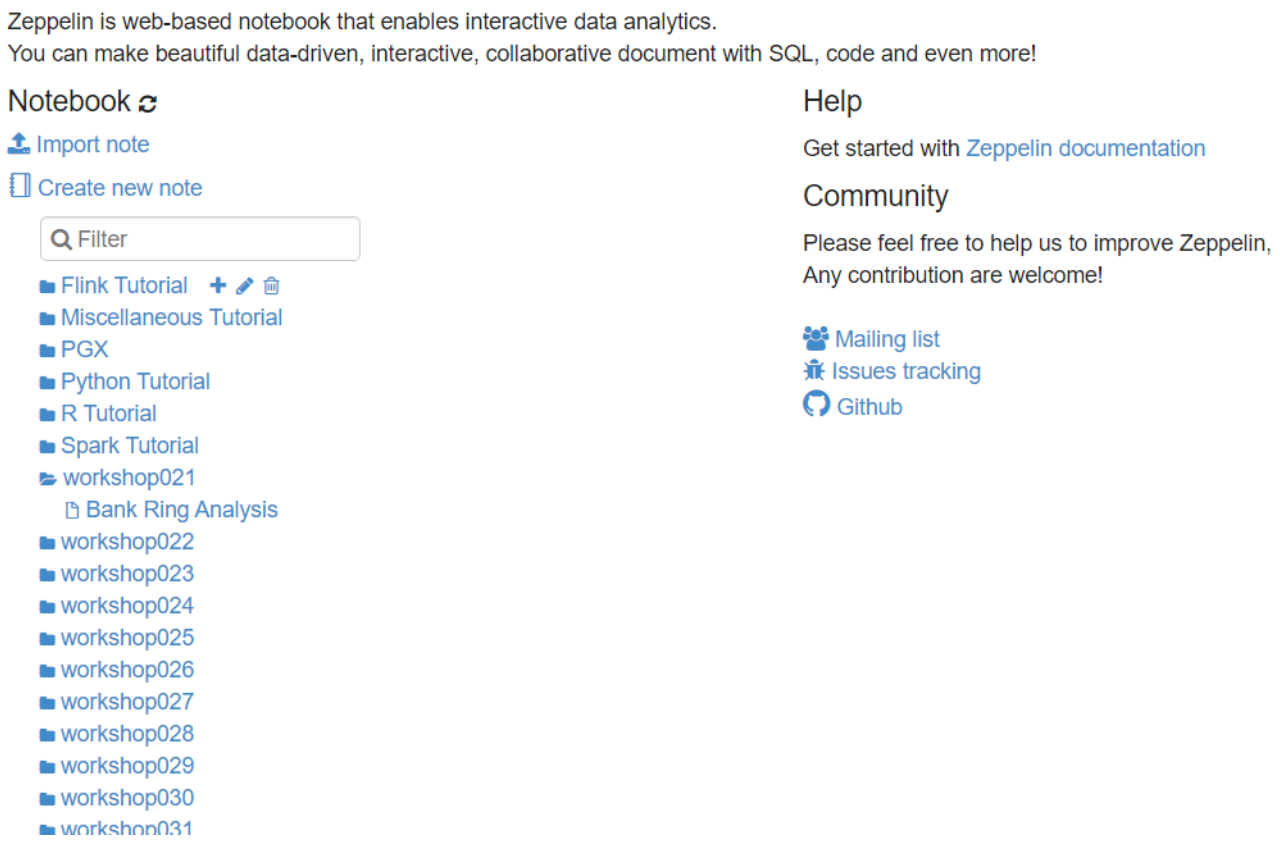
Using the url for your workshop user, you should access to the next Apache Zeppelin page:



Click on “login” button and Use your user “workshopXXX”.



Select the notebook “**Bank Ring Analysis**” in the Folder for your user “workshopXXX”.



Execute each paragraph in this notebook “Step By Step” until the paragraph “First step, look for account sharing contact details: phone number, SSN or address,”.

You can see how you can program in Groovy language or Java Language. The interpreter used by PGX (Property graph Engine) from Oracle Graph is a Groovy/Java interpreter.

To load the graph in Memory from the Graph Data Model, you could use the next code:

```
%pgx

import oracle.pgx.common.types.*

builder.setUsername(user);
builder.setPassword(pass);
// Read the Graph Name created previously
builder.setName("bank");
builder.addVertexProperty("FIRSTNAME", PropertyType.STRING);
builder.addVertexProperty("LASTNAME", PropertyType.STRING);
builder.addVertexProperty("UNIQUEID", PropertyType.STRING);
builder.addVertexProperty("SSN", PropertyType.STRING);
builder.addVertexProperty("PHONENUMBER", PropertyType.STRING);
```

```
builder.addVertexProperty("BALANCE", PropertyType.DOUBLE);
builder.addVertexProperty("APR", PropertyType.DOUBLE);
builder.addVertexProperty("LOANAMOUNT", PropertyType.DOUBLE);
builder.addVertexProperty("STREET", PropertyType.STRING);
builder.addVertexProperty("STATE", PropertyType.STRING);
builder.addVertexProperty("ZIPCODE", PropertyType.STRING);
builder.addVertexProperty("CITY", PropertyType.STRING);
builder.addVertexProperty("ACCOUNTNUMBER", PropertyType.STRING);
builder.addVertexProperty("LIMIT", PropertyType.DOUBLE);
builder.addVertexProperty("SECURITYCODE", PropertyType.STRING);
builder.addVertexProperty("EXPIRATIONDATE", PropertyType.TIMESTAMP);

builder.setLoadVertexLabels(true);
builder.setLoadEdgeLabel(true);
builder.setKeystoreAlias("alias");
```

The name "**bank**" used in this code paragraph is depending on your Graph Data Model, you have created in the Step1.

Step 3: Use PyPGX to access Graph Data Model.

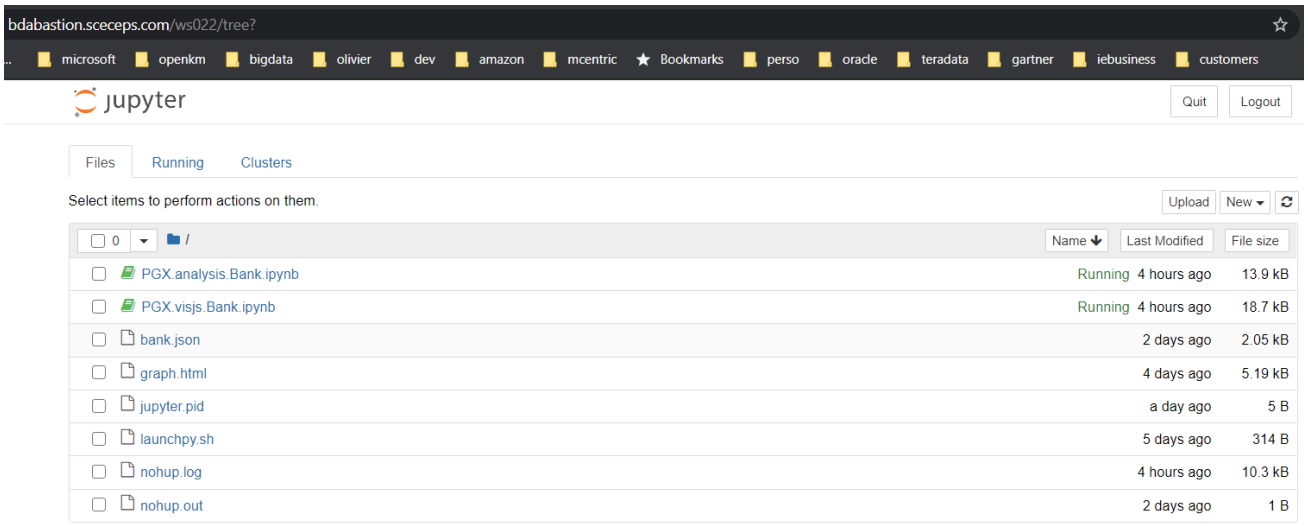
All users "workshopXXX" have been distributed in different Labs:

Users	Jupyter URL
Workshop001 to workshop020	https://bdabastion.sceceps.com/wsXXX
Workshop021 to workshop040	https://bdabastion.sceceps.com/wsXXX
Workshop041 to workshop060	https://bdabastion.sceceps.com/wsXXX
Workshop061 to workshop080	https://bdabastion.sceceps.com/wsXXX
Workshop081 to workshop100	https://bdabastion.sceceps.com/wsXXX

You can find the documentation of PyPGX in the next URL:

https://docs.oracle.com/cd/E56133_01/latest/pythondocs/index.html

With your workshop user, let's connect to https://bdabastion.sceceps.com/wsXXX.



To check how you can access to the Graph Data Model, select the notebook "PGX.analysis.bank.ipynb".

Execute all the different paragraphs of this notebook "step by step".
To load the graph from Oracle Database, we have created a metadata definition "bank.json" where you can choose the vertex and edges you want to load in Memory:

```
{
```

```
"db_engine": "RDBMS",
"vertex_id_type": "long",
"error_handling": {},
"name": "bank",
"vertex_props": [
   {
      "dimension": 0,
      "name": "FIRSTNAME",
      "type": "string"
   },
   {
      "dimension": 0,
      "name": "LASTNAME",
      "type": "string"
   },
   {
      "dimension": 0,
      "name": "UNIQUEID",
      "type": "string"
   },
   {
      "dimension": 0,
      "name": "SSN",
      "type": "string"
   },
   {
      "dimension": 0,
      "name": "PHONENUMBER",
      "type": "string"
   },
   {
      "dimension": 0,
      "name": "BALANCE",
      "type": "double"
   },
   {
      "dimension": 0,
      "name": "APR",
      "type": "double"
   },
   {
      "dimension": 0,
      "name": "LOANAMOUNT",
      "type": "double"
   },
   {
```


```
        "dimension": 0,
        "name": "STREET",
        "type": "string"
    },
    {
        "dimension": 0,
        "name": "STATE",
        "type": "string"
    },
    {
        "dimension": 0,
        "name": "ZIPCODE",
        "type": "string"
    },
    {
        "dimension": 0,
        "name": "CITY",
        "type": "string"
    },
    {
        "dimension": 0,
        "name": "ACCOUNTNUMBER",
        "type": "string"
    },
    {
        "dimension": 0,
        "name": "LIMIT",
        "type": "double"
    },
    {
        "dimension": 0,
        "name": "SECURITYCODE",
        "type": "string"
    },
    {
        "dimension": 0,
        "name": "EXPIRATIONDATE",
        "type": "timestamp"
    }
],
"loading": {
    "load_vertex_labels": true,
    "load_edge_label": true
},
"edge_props": [],
"attributes": {},
```


```
    "format": "pg"
}
```

```
In [4]: print(session)
PgxSession(id: 1a5cadf9-2b36-4729-a6d2-4e10f412e6b1, name: pypgx)
```

Load JSON File with Property Graph Definition

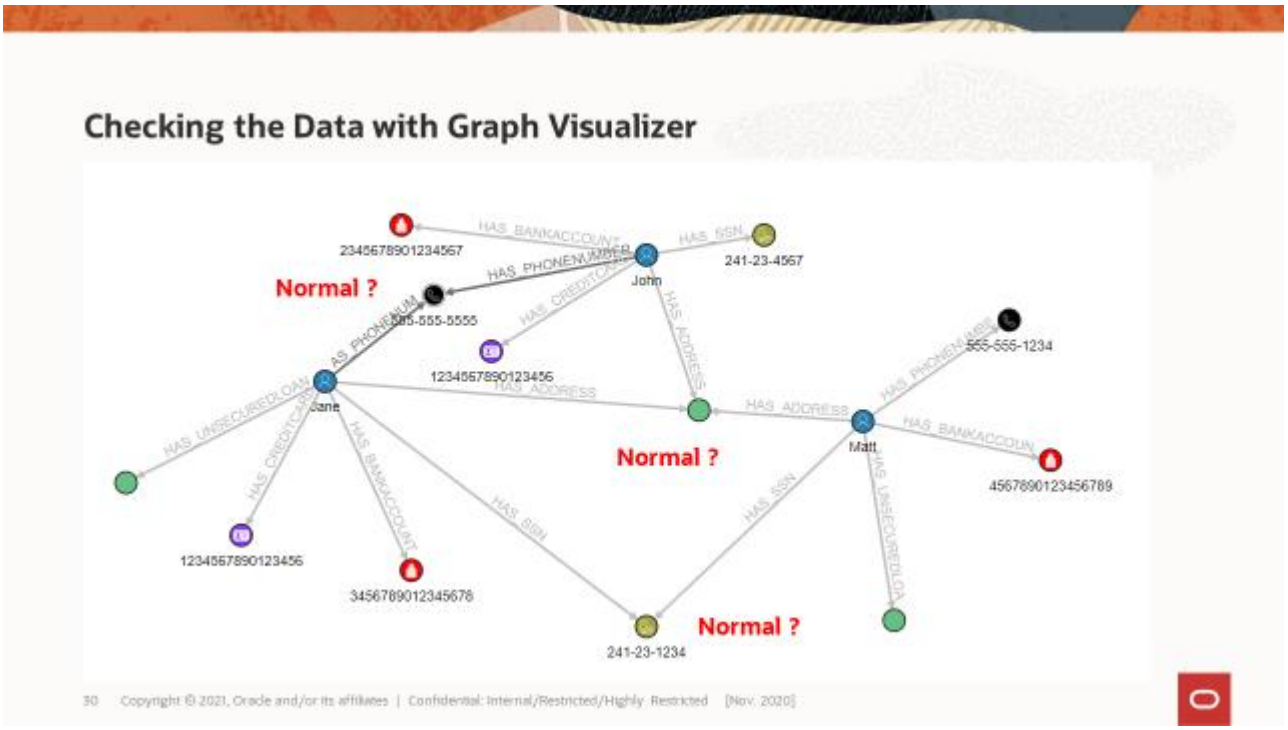
```
In [6]: # read Graph
graph = session.read_graph_with_properties("./bank.json")
```

Now you can compare with the loading mechanism in Apache Zeppelin notebook:

```
%pgx
import oracle.pgx.common.types.*
builder.setUsername(user);
builder.setPassword(pass);
// Read the Graph Name created previously
builder.setName("bank");
builder.addVertexProperty("FIRSTNAME", PropertyType.STRING);
builder.addVertexProperty("LASTNAME", PropertyType.STRING);
builder.addVertexProperty("UNIQUEID", PropertyType.STRING);
builder.addVertexProperty("SSN", PropertyType.STRING);
builder.addVertexProperty("PHONENUMBER", PropertyType.STRING);
builder.addVertexProperty("BALANCE", PropertyType.DOUBLE);
builder.addVertexProperty("APR", PropertyType.DOUBLE);



…etc….
```

Step 4: Use Algorithm your Graph Data Model.

You can check that this Dataset has been prepared to generate some rings in the data:



It is not a clean dataset and now the objective is to use the PGQL query language in order to detect the different rings in the data.

You can analyze the Entities Links between the different Vertex:

Entity Link Analysis

Performing entity link analysis on the above Graph data model is demonstrated below. We use brackets in the below table is to isolate individual elements of a collection.

Find account who share more than one piece of legitimate contact

In [ ]:

```
pgxResultSetNode = graph.query_pgql("""
  select label(contact), listagg(acct.FIRSTNAME,','), count(*) as size
from match (acct)-[]->(contact)
where exists (
  SELECT count(*) as ringsize, contact
  from MATCH (account:CUST)-[]->(contact)
  group by contact
  having ringsize > 1
  order by ringsize desc
)
group by label(contact)
""")

for i in pgxResultSetNode:
    print (i)
```

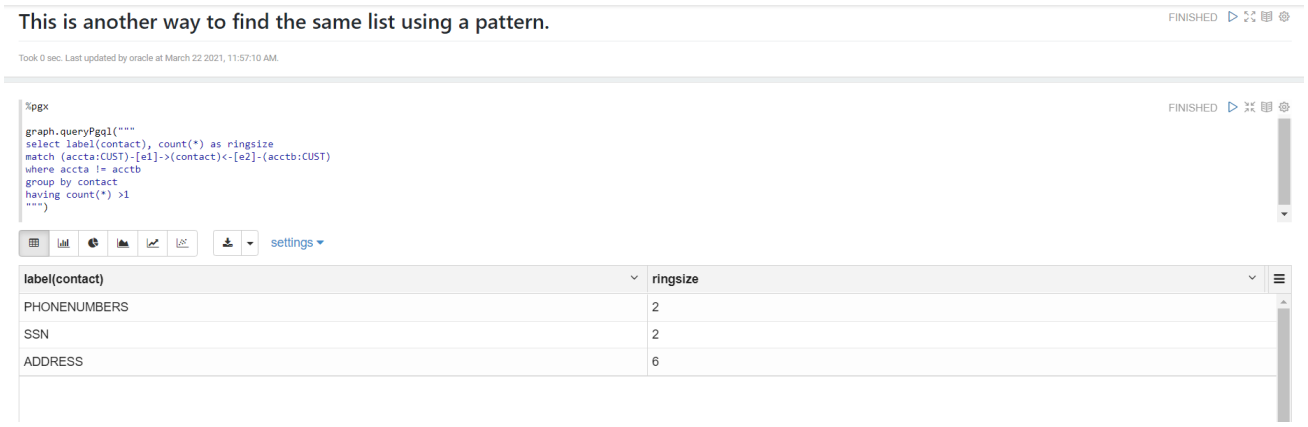
You can check the Financial Risk for possible Fraud Ring (PyPGX).

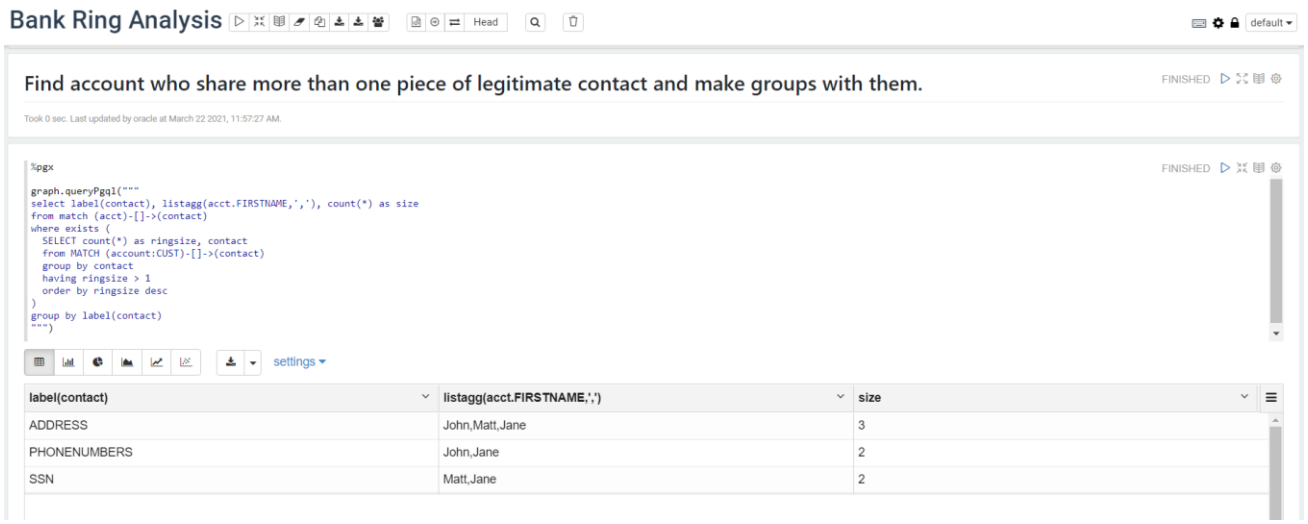
Determine the financial risk of a possible fraud ring

In [ ]:

```
pgxResultSetNode = graph.query_pgql("""
select label(contact), listagg(acct.FIRSTNAME,','),
       count(*) as size, round(sum(CASE
                                    WHEN label(r)='HAS_CREDITCARDS' THEN unsecuredAccount.LIMIT
                                    WHEN label(r)='HAS_UNSECUREDLOANS' THEN unsecuredAccount.BALANCE
                                    ELSE 0
                                   END)) as FINANCIALRISK
from match (acct)-[]->(contact),
     match (acct)-[r:HAS_CREDITCARDS|HAS_UNSECUREDLOANS]->(unsecuredAccount)
where exists (
  SELECT count(*) as ringsize, contact
  from MATCH (account:CUST)-[]->(contact)
  group by contact
  having ringsize > 1
  order by ringsize desc
)
group by label(contact)
""")

for i in pgxResultSetNode:
    print (i)
```

You can compare with Apache Zeppelin Notebook execution using the Groovy language:

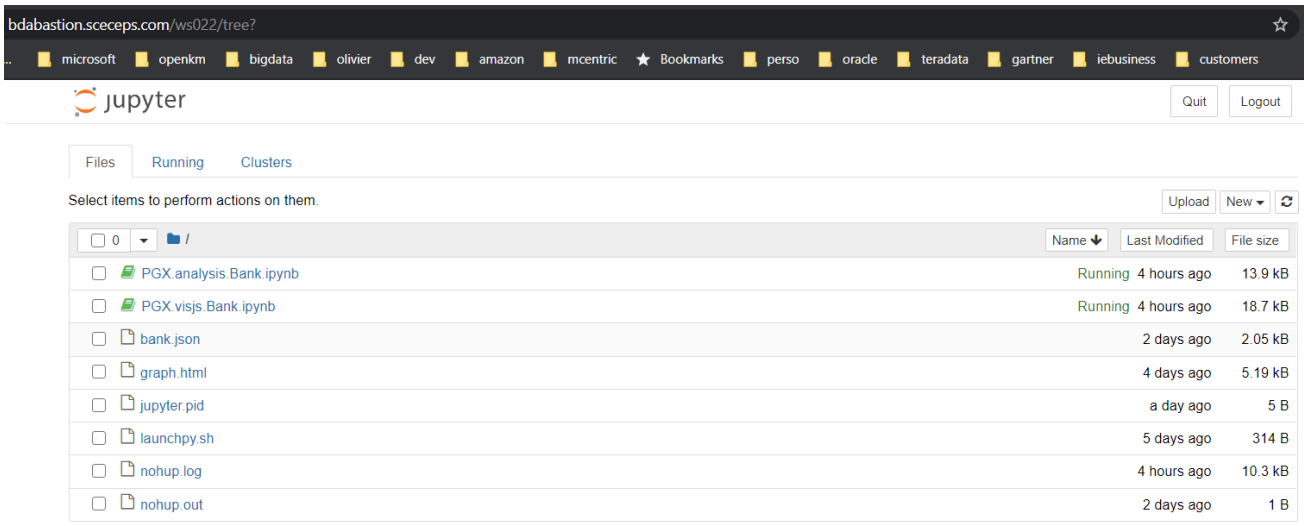




Step 5: Use Open Source Vis.JS to visualize your Graph Data Model.

Connect to Jupyter Notebook using your workshop user (change the XXX by your userid 001 to 100):

https://bdabastion.sceceps.com/wsXXX



Execute the notebook "**PGX.visjs.Bank.ipynb**".

In this notebook, we are using the Python Library “pyvis” where the open source JavaScript Library Vis.JS has been embedded.

https://visjs.org/

In this open source library, a widget “Network” has been implemented in order to represent a graph in JavaScript component.

https://visjs.github.io/vis-network/docs/network/

Execute Queries in order to create Nodes List and Edges List for Vis.JS:

Query to prepare Data for Vis.JS, Network and NetworkX (Python)

In [ ]:

```
# get result
# node data
pgxResultSetNode = graph.query_pgql("""
  SELECT id(x), label(x), x.FIRST_NAME, x.CITY
  MATCH (x)-[]-()
""")

#['size', 'value', 'title', 'x', 'y', 'label', 'color']

node_id = []
node_title =[]
node_value = []
node_label = []

for i in pgxResultSetNode:
    size = i[0]
    if size not in node_id:
        node_id.append(size)
        node_label.append(i[1])
        if i[2] != '':
           node_title.append(i[2])
           print('Node: size: ' + str(i[0]) + ' label: ' + i[1] + ' FIRST_NAME: ' + i[2])
        elif i[3] != '':
           node_title.append(i[3])
           print('Node: size: ' + str(i[0]) + ' label: ' + i[1] + ' CITY: ' + i[3])
        else:
           node_title.append('n/a')
           print('Node: size: ' + str(i[0]) + ' label: ' + i[1] + ' title: n/a')
```

Using Network:

Open Source Vis.JS Visualization using Network Python Library

In [ ]:

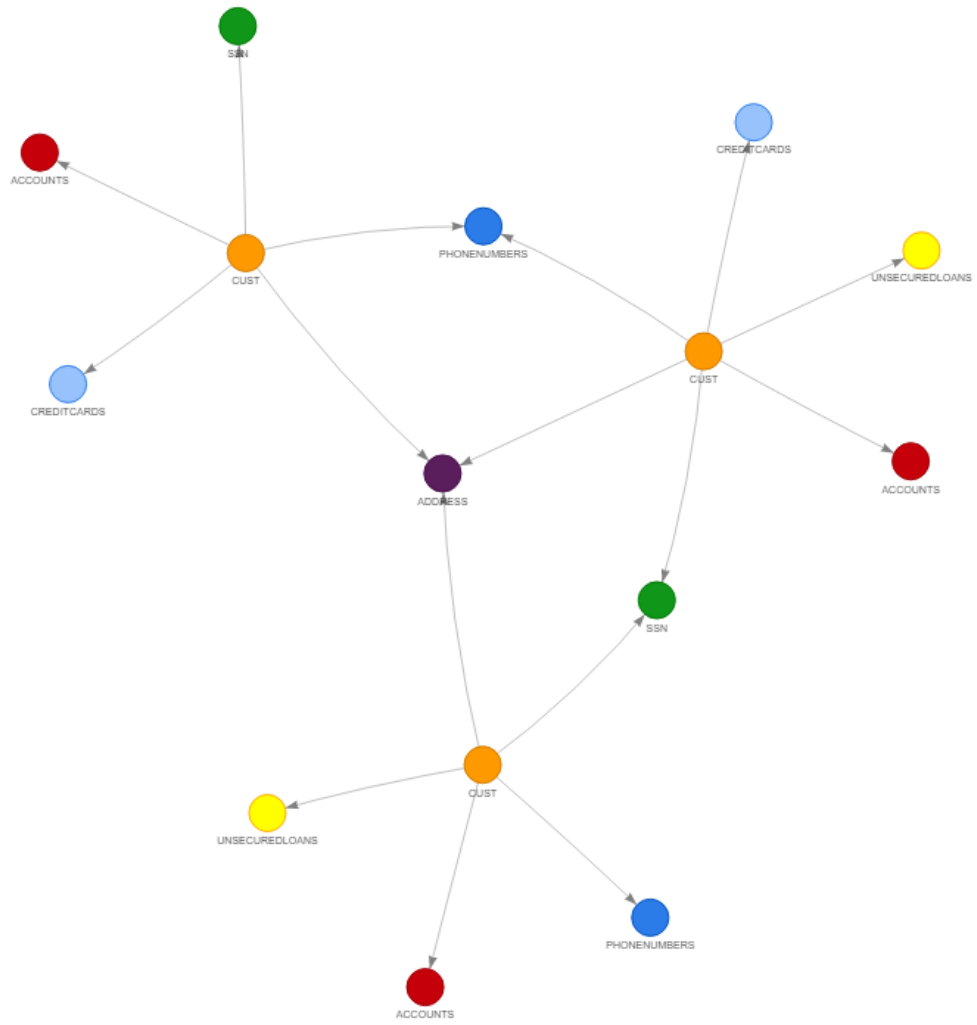
```
from pyvis.network import Network

g = Network(notebook=True, height = '800px', width = '100%' , directed = True)
g.options = {
          "nodes": {
            "scaling": {
              "min": 16,
              "max": 32,
            },
          },
          "edges": {
            "color": "GRAY",
            "smooth": "false",
          },
          "physics": {
            "barnesHut": { "gravitationalConstant": -30000 },
            "stabilization": { "iterations": 2500 },
          }
}

g.add_nodes(node_id,label=node_label,title=node_title)
g.add_edges(edge_list)
#g.show_buttons()
g.show('graph.html')
```

Using NetworkX:

Another Vis.JS Visualization using NetworkX Python Library

In [ ]:

```
from pyvis.network import Network
import networkx as nx

nx_graph = nx.Graph()

pgxResultSetNode = graph.query_pgql("""
  SELECT id(x), label(x), x.FIRSTNAME, x.CITY
  MATCH (x)-[]-()
""")

for i in pgxResultSetNode:
    if i[2]:
      nx_graph.add_node(i[0], size=20, title=i[2], group=i[1])
    else:
      nx_graph.add_node(i[0], size=20, title=i[3], group=i[1])

# edge data
pgxResultSetEdge = graph.query_pgql("""
  SELECT id(x), id(y), label(e)
  MATCH (x)-[e]->(y)
""")

edge_list = []
for i in pgxResultSetEdge:
    nx_graph.add_edge(i[0], i[1], weight=5)

nt = Network(notebook=True, height = '800px', width = '100%')
# populates the nodes and edges data structures
nt.from_nx(nx_graph)
nt.show('nx.html')
```



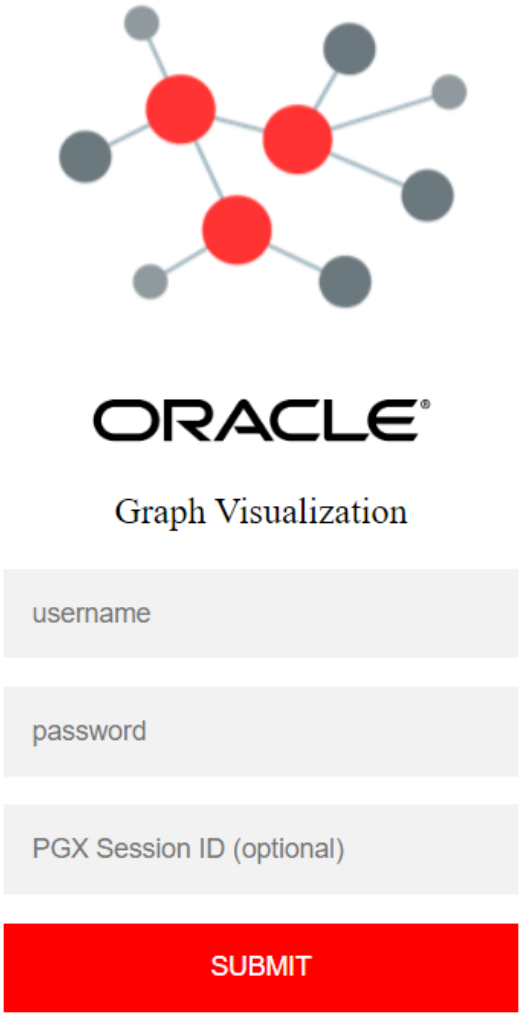
Step 6: Use Graph UI to visualize your Graph Data Model.

Oracle Graph includes an UI Visualizer for PGX using JavaScript libraries.

You can access to this tool using the next URL:
https://bdabastion.sceceps.com/ui

Use the next generic information to login to the tool “PGX Visualizer”:

User: bankuser
Pwd: welcome1



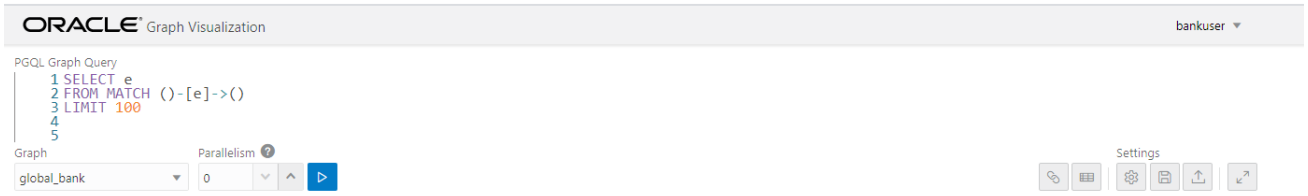
Select the “global_bank” Graph Name in the combobox.

Copy the next query to select all relationships (edges):

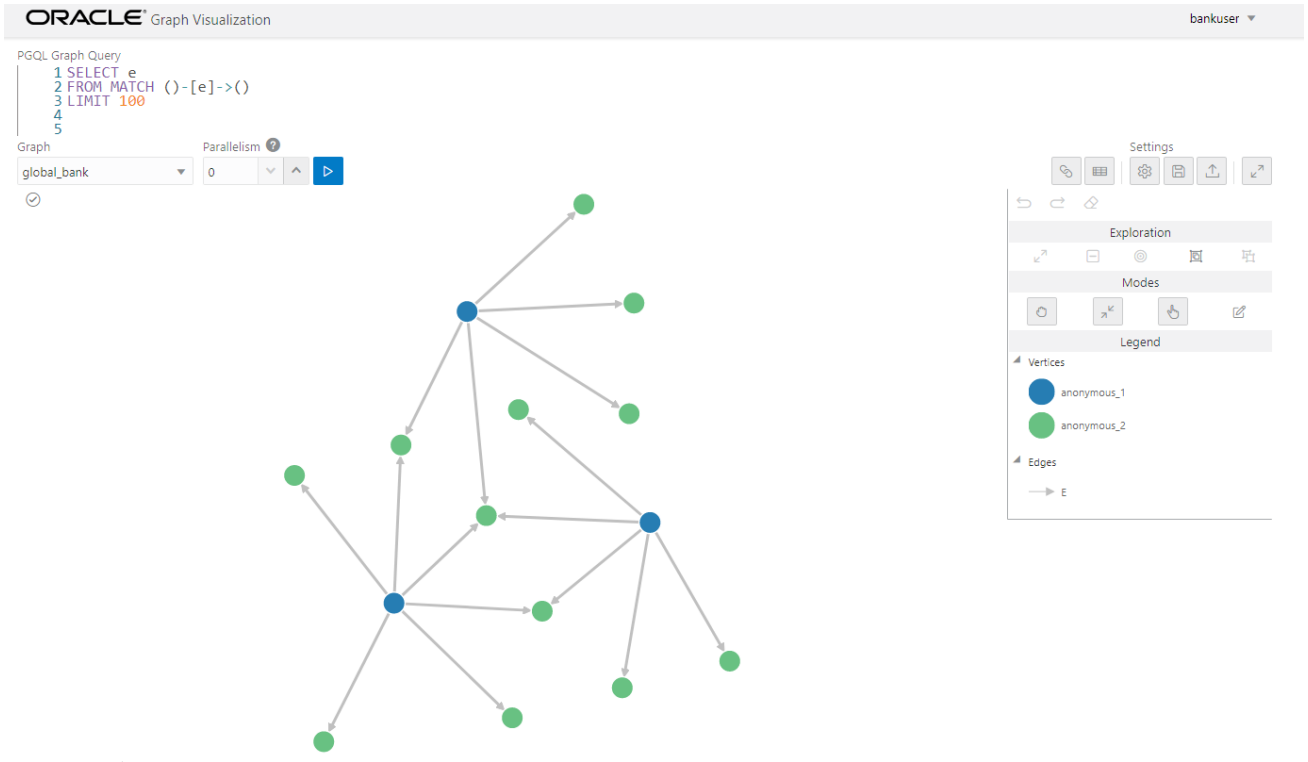
```
SELECT e
FROM MATCH ()-[e]->()
LIMIT 100
```

In order to draw a visualization, you can limit the number of results.

Execute the next button “Run Query”:



The PGX Visualizer using a Javascript library draws the Vertex and Edges list in the UI:



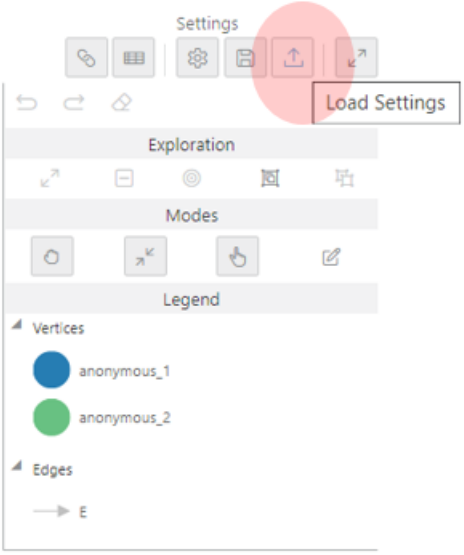
By default, the PGX Visualizer generate a visualization for Vertex and Edges and you can customize the visualization using the toolbar button on the right side.

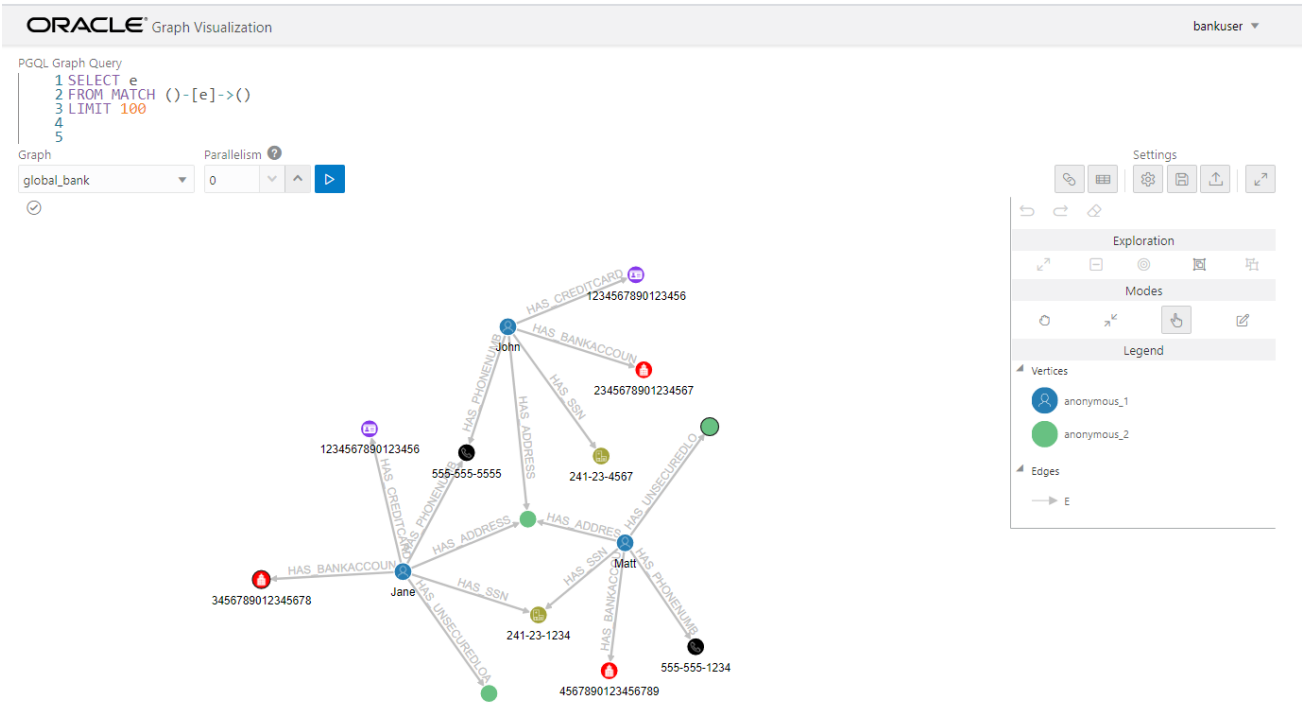
You can download the next JSON Configuration file from github repository “https://github.com/operard/mlgraph/blob/main/graphviz/bank_settings_20210315.json”

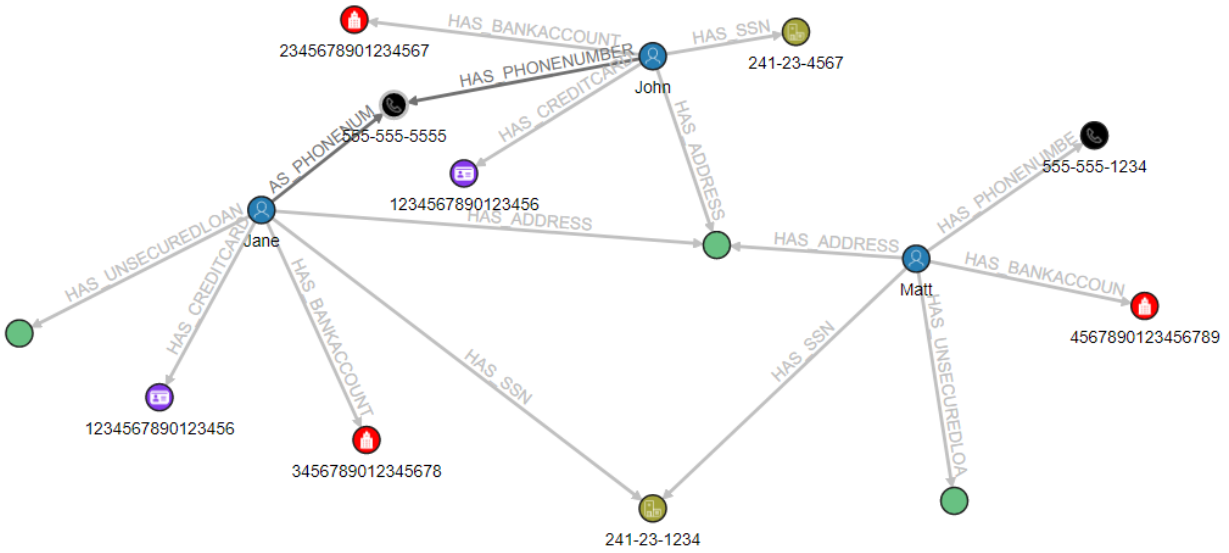
{"version":6,"theme":"light","height":300,"similarEdges":"keep","edgeMarker":"arrow","vertexGeometry":"sphere","antiAlias":true,"floorGrid":true,"edgesGeometry":true,"enable3DLayout":false,"selectedPropZ":"","valueRangePropZ":0.5,"sortPropZ":"descending","filters":[{"_id":1615824847813,"type":"styling","enabled":true,"conditions":{"operator":"and","conditions":[{"property":"label","operator":"=","value":"CUST"}]},"component":"vertex","target":"vertex","properties":{"icons":["fa-user"],"iconColors":["white"],"label":["FIRSTNAME"]}},{"_id":1615824870065,"type":"styling","enabled":true,"conditions":{"operator":"and","conditions":[]},"component":"edge","target":"edge","properties":{"label":["label"]}},{"_id":1615824919274,"type":"styling","enabled":true,"conditions":{"operator":"and","conditions":[{"property":"label","operator":"=","value":"PHONENUMBERS"}]},"component":"vertex","target":"vertex","properties":{"colors":["black"],"icons":["fa-phone"],"iconColors":["white"],"label":["PHONENUMBER"]}},{"_id":1615828536887,"type":"styling","enabled":true,"conditions":{"operator":"and","conditions":[{"property":"label","operator":"=","value":"ACCOUNTS"}]},"component":"vertex","target":"vertex","properties":{"colors":["red"],"icons":["fa-building"],"iconColors":["white"],"label":["ACCOUNTNUMBER"]}},{"_id":1615828630015,"type":"styling","enabled":true,"conditions":{"operator":"and","conditions":[{"property":"label","operator":"=","value":"SSN"}]},"component":"vertex","target":"vertex","properties":{"colors":["rgb(163, 163, 57)"],"icons":["fa-hospital-o"],"iconColors":["white"],"label":["SSN"]}},{"_id":1615828721839,"type":"styling","enabled":true,"conditions":{"operator":"and","conditions":[{"property":"label","operator":"=","value":"CREDITCARDS"}]},"component":"vertex","target":"vertex","properties":{"colors":["rgb(134, 56, 236)"],"icons":["fa-id-card"],"iconColors":["white"],"label":["ACCOUNTNUMBER"]}}],"smartExpands":[],"smartGroups":[],"vertexLabelProperty":null,"edgeLabelProperty":null,"vertexLabelOrientation":"bottom","vertexPositions":[{"id":"6915093083595417275","x":0.39905679562582147,"y":0.418391158327987},{"id":"565234722105082376","x":0.3067477945341144,"y":0.5443462008779054},{"id":"3824048412016813344","x":0.4270791709572325,"y":0.557312161140396 9},{"id":"6696452419939143867","x":0.7238514251596001,"y":0.3256123683876622},{"id":"524214483973154798","x":0.3601054994781409,"y":0.5830259232993977},{"id":"3265555656079332997","x":0.4066149976091672,"y":0.2959708075190381},{"id":"7012961393350504935","x":0.5648861255774357,"y":0.7062724132268348},{"id":"4090088222170435427","x":0.43924106346227515,"y":0.5299831301660705},{"id":"879672803699099970","x":0.21502363217389478,"y":0.4129769688425543},{"id":"2074598160484072345","x":0.26778067482998424,"y":0.614227566319002},{"id":"6669874926900192849","x":0.5662744688052275,"y":0.38801565442687086},{"id":"7396233240736154172","x":0.32262023232776155,"y":0.7858366029268257},{"id":"8564415915979133027","x":0.19419848375701743,"y":0.22732719287590866},{"id":"5163095964725049 86","x":0.5586385810523725,"y":0.15400333177983858},{"id":"6180573890730454984","x":0.839778084680218,"y":0.19612554985630437},{"id":"6840400127837741695","x":0.8619915763248872,"y":0.4582193512209805},{"id":"3483432232742040934","x":0.7849385271824407,"y":0.6953518381699731},{"id":"6281001637277489523","x":0.6655410095923434,"y":0.7624353706621224}],"pageSize":100,"selectedPage":0,"charLimit":10,"truncateLabel":false,"showTitle":true,"dimension":"2d","layout":"force","layoutProperties":{"circle":{"avoidOverlap":true,"radius":40},"grid":{"avoidOverlap":true,"rows":5,"colu

mns":5},"concentric":{"avoidOverlap":true,"minimumVertexSpacing":10},"force":{"edgeDistance":120,"forceStrength":-30,"vertexPadding":40,"velocityDecay":0.3},"force3D":{"attractionMultiplier":20,"repulsionMultiplier":0.2},"geographical":{"latitude":"latitude","longitude":"longitude"},"hierarchical":{"rankDirection":"TB","ranker":"network-simplex","align":"","vertexSeparation":{"value":0,"enabled":false},"edgeSeparation":{"value":0,"enabled":false},"rankSeparation":{"value":0,"enabled":false}},"radial":{"arcDegree":360,"startingPoint":"left","packing":1,"intelligentSeparation":true}},"animateChanges":true,"liveSearch":{"enabled":false,"enabledTypes":[],"selectedColumns":[],"value":"","location":0,"distance":100,"maxPatternLength":32,"minMatchCharLength":1,"advancedSettingsToggle":false,"disabledSelectedColumns":[]},"graphAction":{"enabled":true,"graphName":"bank_21","lastRun":0,"metadata":{"vertexProperties":[],"edgeProperties":[]},"stack":[],"index":0,"numberOfHops":1,"smartExpand":0,"smartGroup":0},"interactionMode":"zoom","networkEvolution":{"enabled":false,"vertexProperty":"id","vertexEndProperty":"","dataType":"int","axis":"vertices","overview":"on","height":100,"overviewChartType":"bar","valuesToExclude":[],"valuesToExcludeType":"hideBoth","stepSize":0,"edgeProperty":"id","edgeEndProperty":"","playTimeout":1000,"playStep":1,"advancedSettingsToggle":false},"stickyVertices":true,"annotationMode":false,"fitToScreen":false,"viewTransform":[[-220.75572591785055,-71.71488606101718,1.33262346325533,1081]],"showLegend":true,"visibleGraphMode":{"enabled":false,"name":""},"width":100}

You can upload this JSON configuration file “bank_settings_20210315.json” using the “Load Settings” button.







Conclusions

In this workshop, we have seen how you can reuse and convert a 3FN relational data model in order to apply Queries and Machine Learning algorithms using the same Oracle Database.

This functionality is available in order to apply fast use cases on the relationships between entities.

You can continue to implement Oracle Graph use cases using the free labs available in the next URLs:

- Oracle Property Graph for Real-Time Recommendations Workshop

https://apexapps.oracle.com/pls/apex/dbpm/r/livelabs/view-workshop?wid=754

- Analyze and Visualize Property Graphs with Oracle Database Workshop

https://apexapps.oracle.com/pls/apex/dbpm/r/livelabs/view-workshop?wid=687

- Getting started with Oracle Property Graph on Docker Workshop

https://apexapps.oracle.com/pls/apex/dbpm/r/livelabs/view-workshop?wid=712

Other Labs in Machine Learning, AI, Data Management, …etc… are available in the next URL:

https://apexapps.oracle.com/pls/apex/dbpm/r/livelabs/livelabs-workshop-cards?clear=100&session=101956397438991