Parallel Graph AnalytiX (PGX)

In the following set of notebooks you can find code to execute simple graph operations using as key component the Parallel Graph AnalytiX (PGX) graph analysis toolkit.

There are few ways to interact with PGX:

- interactive shell
- PGX client
- REST api

The *interactive shell* can be started by executing `$ORACLE_HOME/md/property_graph/pgx/bin/pgx` . Most of the code used in the following notebooks will work in the shell with minimal changes (needs to remove the code required by python to interact with java). Obviously visualizations and pure python code can't used in the shell, but there is full access to PGX methods.
The *PGX client* is mainly a layer on top of the *REST api* when connecting to a remote PGX instance, while acting like the *interactive shell* when executed in local mode. Locally it can be embedded into exiting java applications to have graph databases.
When using the *PGX client* to connect to a remote PGX instance it will take care of all the *REST api* interactions. Worth nothing that most of REST api works in an asynchronous way: the one doing a call will need to go back again and again asking if the job is done. The *PGX client* take care of that providing blocking methods which will stop the executing and wait till the job on the server is done.
More details on the various ways to use PGX can be found on https://docs.oracle.com/cd/E56133_01/latest/reference/overview/usage.html (https://docs.oracle.com/cd/E56133_01/latest/reference/overview/usage.html).

Requirements & Environment

This notebook make use of *PGX 3.1.1* which is "embedded" into Oracle Database. It is available as a patch to update the out-of-the-box version (Patch 28577866: MISSING LATEST PGX 3.1.X FUNCTIONS IN ORACLE 12.2 AND 18.1).
The PGX server is configured to listen on *port 7007* without SSL and authentication is disabled (these settings are defined into *conf/server.conf*). Access to the host local filesystem has also been enabled in the *conf/pgx.conf* file to allow local storage and loading of a graph from a remote client.

PGX can be found in `$ORACLE_HOME/md/property_graph/pgx` . The libraries required by the client connecting to the server are stored in `$ORACLE_HOME/md/property_graph/lib` .
The PGX server needs to be started by executing `$ORACLE_HOME/md/property_graph/pgx/bin/start-server` (or deployed instead as a webapp in an application server like Weblogic).

This Notebook is using python 3.6, but the graph code works on python 2.7 too. *JPype1* is required and can be installed using `pip install JPype1` (this package is the one connecting python with the JVM where the PGX java commands will be executed). graphviz is used to display the sample graph and can be installed using `pip install graphviz` too, this method will not work with huge graphs because the resulting image would be too big. pandas is used to easily manipulate, analyze and visualize data structures in python and can also be installed using `pip install pandas` .
Alternative

Notebooks in this serie

- 1 Graph by hand (1%20Graph%20by%20hand.ipynb)

- 2 Create graph in database (2%20Create%20graph%20in%20database.ipynb)
- 3 Load graph from database (3%20Load%20graph%20from%20database.ipynb)
- 4 Save and load graph as file (4%20Save%20and%20load%20graph%20as%20file.ipynb)
- 5 Convert PGQL to SQL (5%20Convert%20PGQL%20to%20SQL.ipynb)

Create a graph by hand

A graph can be built by hand simply defining nodes and edges one by one with all their properties and labels.

Import required packages

In [1]:

```
from jpype import *
import os
```

Setup JVM

Start JVM passing the PGX 3.1.1 classpath.
getDefaultJVMPath() will try to detect the location of java in the system, it can be replaced by the string defining the path to `libjvm.so` .

In [2]:

```
# %load -s _get_pgx_class_path ../graphUtils.py
def _get_pgx_class_path(pgx_directory):
    class_path_list = []
    for root, dirs, files in os.walk(pgx_directory):
        for file in files:
            if file.endswith('.jar'):
                class_path_list.append(os.path.join(root, file))
    return ':'.join(class_path_list)
```

In [3]:

```
pgxLibsPath = '/opt/oracle/product/18c/dbhome_1/md/property_graph/lib'
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path(pgxLibsPa
th))
```

Create a session on the PGX 3.1.1 server

Need to start the PGX 3.1.1 server before to continue:

```
$ORACLE_HOME/md/property_graph/pgx/bin/start-server
```

In [4]:

```
session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7007/", "my_sess
ion")
```

Create a new graph

In [5]:

```
# new builder
builder = session.newGraphBuilder(JClass('oracle.pgx.common.types.IdType').LONG)

# define some nodes (node ID is unique!)
builder.addVertex(1).addLabel("person").setProperty("name", "Francesco").setProperty("c
ountry", "Italy")
builder.addVertex(2).addLabel("person").setProperty("name", "Christian").setProperty("c
ountry", "Switzerland")
builder.addVertex(3).addLabel("session").setProperty("name", "Starting an Oracle Analyt
ics Cloud Journey from 0")
builder.addVertex(4).addLabel("event").setProperty("name", "ITOUG 2019")
builder.addVertex(5).addLabel("country").setProperty("name", "Italy")
builder.addVertex(6).addLabel("country").setProperty("name", "Switzerland")
builder.addVertex(7).addLabel("continent").setProperty("name", "Europe")

# define some edges (edge ID is unique!)
builder.addEdge(0, 1, 2).setLabel("friendOf")
builder.addEdge(1, 1, 3).setLabel("presents")
builder.addEdge(2, 2, 3).setLabel("presents")
builder.addEdge(3, 3, 4).setLabel("scheduledAt")
builder.addEdge(4, 1, 5).setLabel("livesIn")
builder.addEdge(5, 2, 6).setLabel("livesIn")
builder.addEdge(6, 5, 7).setLabel("partOf")
builder.addEdge(7, 6, 7).setLabel("partOf")
builder.addEdge(8, 4, 5).setLabel("happensIn")
```

Out[5]:

```
<jpype._jclass.oracle.pgx.api.graphbuilder.EdgeBuilderImpl at 0x7f76c821ac
f8>
```

Build new graph

In [6]:

```
graph = builder.build()

print(graph)
```

```
PgxGraph[name=anonymous_graph_3,N=7,E=9,created=1551456729641]
```

Visualize the graph

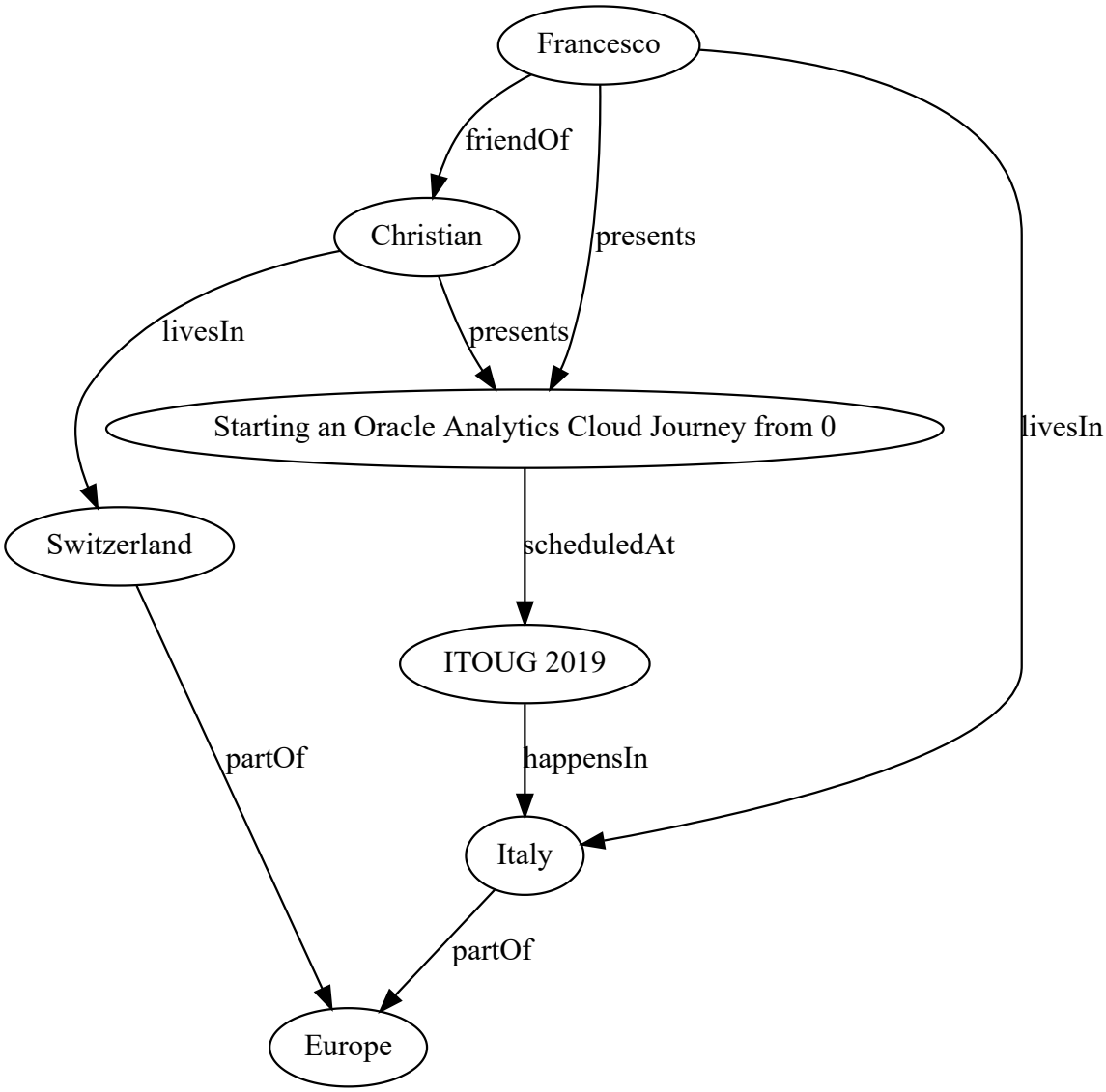
In [7]:

```
# %load -s renderGraph ../graphUtils.py
def renderGraph(graph):
    from graphviz import Digraph

    # get all the vertices of the graph
    vertices = graph.getVertices()
    # create a new visualization
    dot = Digraph(comment='Graph')
    # loop over vertices
    for v in vertices.iterator():
        dot.node(str(v.getId()), v.getProperty("name"))

    # loop over vertices to get 'out' edges
    for v in vertices.iterator():
        edges = v.getOutEdges()
        # loop over 'out' edges
        for e in edges:
            dot.edge(str(e.getSource().getId()), str(e.getDestination().getId()), label
=e.getLabel())

    # return (display) graph
    return dot
```

In [8]:

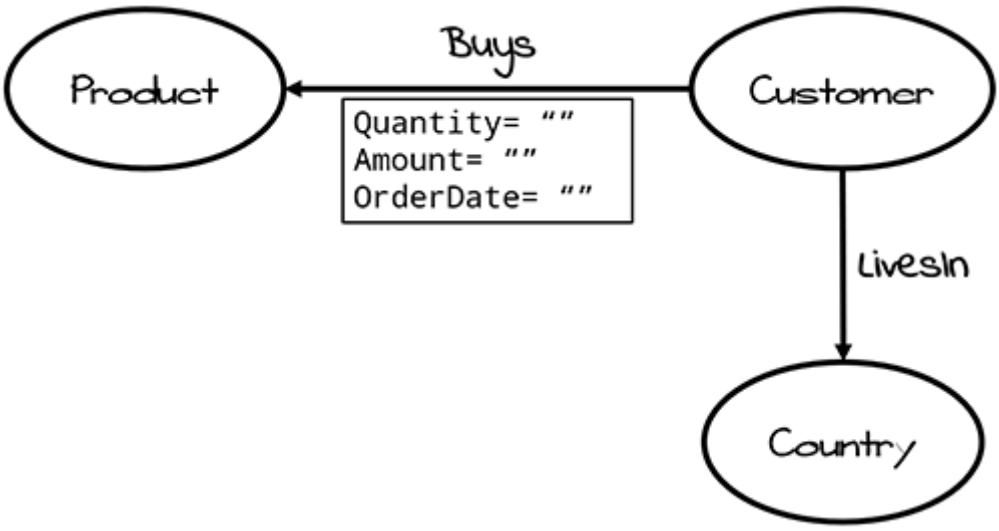
```
renderGraph(graph)
```

Out[8]:


Francesco
friendOf
Christian
presents
livesIn
presents
Starting an Oracle Analytics Cloud Journey from 0
livesIn
Switzerland
scheduledAt
ITOUG 2019
partOf
happensIn
Italy
partOf
Europe

Create a graph in the database

Using the SH sample schema as base (available on https://github.com/oracle/db-sample-schemas (https://github.com/oracle/db-sample-schemas)), taking *customers*, *countries*, *products* and *sales* as source tables for the graph.



Create a new empty graph in the database

Call a package method, all the available methods and their properties can be found at https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/OPG_APIS-reference.html (https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/OPG_APIS-reference.html) This is equivalent to

```
BEGIN
    OPG_APIS.CREATE_PG('mysales', 4, 8, '');
END;
```

In [1]:

```
import cx_Oracle
con = cx_Oracle.connect('scott/Admin123@localhost:1521/ORCLPDB1')
cur = con.cursor()
cur.callproc('OPG_APIS.CREATE_PG', ['mysales', 4, 8, ''])
cur.close()
con.close()
```

In [2]:

```
%load_ext sql
%sql oracle://scott:Admin123@localhost:1521/?service_name=ORCLPDB1
```

Out[2]:

```
'Connected: scott@'
```

In [3]:

```
%%sql
SELECT owner, table_name
FROM all_tables
WHERE owner = 'SCOTT'
AND table_name like 'MYSALES%'
ORDER BY table_name
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[3]:

owner	table_name
SCOTT	MYSALESGE$
SCOTT	MYSALESGT$
SCOTT	MYSALESIT$
SCOTT	MYSALESSS$
SCOTT	MYSALESVT$

Inspect source data to define IDs and properties

In [4]:

```
%%sql
SELECT 'customer ID' as id, MIN(cust_id) as min_id, MAX(cust_id) as max_id, COUNT(DISTI
NCT cust_id) as unique_id, COUNT(*) as nrows FROM sh.customers
UNION ALL
SELECT 'product ID', MIN(prod_id), MAX(prod_id), COUNT(DISTINCT prod_id) as unique_id,
 COUNT(*) as nrows FROM sh.products
UNION ALL
SELECT 'country ID', MIN(country_id), MAX(country_id), COUNT(DISTINCT country_id) as un
ique_id, COUNT(*) as nrows FROM sh.countries
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[4]:

id	min_id	max_id	unique_id	nrows
customer ID	1	104500	55500	55500
product ID	13	148	72	72
country ID	52769	52791	23	23

There are potential overlaps in IDs of the 3 tables, but rows are unique by ID.
A solution could be to use a sequence to make sure to have uniques IDs for vertices.
In this case a "shortcut" (cheat/workaround) will be used to make sure there is no overlap, simply by adding a fixed number to each ID of the *products* and *countries* tables.

In [5]:

```
%%sql
SELECT 'customer ID' as id, MIN(cust_id) as min_id, MAX(cust_id) as max_id FROM sh.cust
omers
UNION ALL
SELECT 'product ID', MIN(prod_id + 200000), MAX(prod_id + 200000) FROM sh.products
UNION ALL
SELECT 'country ID', MIN(country_id + 300000), MAX(country_id + 300000) FROM sh.countri
es
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[5]:

id	min_id	max_id
customer ID	1	104500
product ID	200013	200148
country ID	352769	352791

Create nodes

The **T** column is a value representing the data type (ref. https://docs.oracle.com/cd/E56133_01/latest/reference/loader/file-system/plain-text-formats.html (https://docs.oracle.com/cd/E56133_01/latest/reference/loader/file-system/plain-text-formats.html))

1) Countries

Data by row

In [6]:

```
%%sql result <<
SELECT country_id + 300000 as vid
, 'label' as k
, 1 as t
, 'country' as v
, null as vn FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid
, 'name' as k
, 1 as t
, country_name as v
, null as vn FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid
, 'isoCode' as k
, 1 as t
, country_iso_code as v
, null as vn FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid
, 'sourceId' as k
, 2 as t
, TO_CHAR(country_id) as v
, country_id as vn FROM sh.countries
ORDER BY 1,2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
Returning data to local variable result
```

In [7]:

```
import pandas as pd

result_df = result.DataFrame()
result_df.head(8)
```

Out[7]:

	vid	k	t	v	vn
0	352769	isoCode	1	SG	NaN
1	352769	label	1	country	NaN
2	352769	name	1	Singapore	NaN
3	352769	sourceId	2	52769	52769.0
4	352770	isoCode	1	IT	NaN
5	352770	label	1	country	NaN
6	352770	name	1	Italy	NaN
7	352770	sourceId	2	52770	52770.0

Insert rows in MYSALESVT$

Because there isn't any date value, the 'vt' column isn't defined

In [8]:

```
%%sql
INSERT INTO MYSALESVT$ (vid, k, t, v, vn)
SELECT country_id + 300000 as vid
, 'label' as k
, 1 as t
, 'country' as v
, null as vn FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid
, 'name' as k
, 1 as t
, country_name as v
, null as vn FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid
, 'isoCode' as k
, 1 as t
, country_iso_code as v
, null as vn FROM sh.countries
UNION ALL
SELECT country_id + 300000 as vid
, 'sourceId' as k
, 2 as t
, TO_CHAR(country_id) as v
, country_id as vn FROM sh.countries
ORDER BY 1,2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
92 rows affected.
```

Out[8]:

```
[]
```

2) Products

Data by row

In [9]:

```
%%sql result <<
SELECT prod_id + 200000 as vid
, 'label' as k
, 1 as t
, 'product' as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'name' as k
, 1 as t
, prod_name as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'category' as k
, 1 as t
, prod_category as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'subcategory' as k
, 1 as t
, prod_subcategory as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'listPrice' as k
, 3 as t
, TO_CHAR(prod_list_price) as v
, prod_list_price as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'sourceId' as k
, 2 as t
, TO_CHAR(prod_id) as v
, prod_id as vn FROM sh.products
ORDER BY 1,2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
Returning data to local variable result
```

In [10]:

```
result_df = result.DataFrame()
result_df.head(12)
```

Out[10]:

	vid	k	t	v	vn
0	200013	category	1	Photo	None
1	200013	label	1	product	None
2	200013	listPrice	3	899.99	899.99
3	200013	name	1	5MP Telephoto Digital Camera	None
4	200013	sourceId	2	13	13
5	200013	subcategory	1	Cameras	None
6	200014	category	1	Peripherals and Accessories	None
7	200014	label	1	product	None
8	200014	listPrice	3	999.99	999.99
9	200014	name	1	17" LCD w/built-in HDTV Tuner	None
10	200014	sourceId	2	14	14
11	200014	subcategory	1	Monitors	None

Insert rows in MYSALESVT$

Because there isn't any date value, the 'vt' column isn't defined

In [11]:

```
%%sql
INSERT INTO MYSALESVT$ (vid, k, t, v, vn)
SELECT prod_id + 200000 as vid
, 'label' as k
, 1 as t
, 'product' as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'name' as k
, 1 as t
, prod_name as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'category' as k
, 1 as t
, prod_category as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'subcategory' as k
, 1 as t
, prod_subcategory as v
, null as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'listPrice' as k
, 3 as t
, TO_CHAR(prod_list_price) as v
, prod_list_price as vn FROM sh.products
UNION ALL
SELECT prod_id + 200000 as vid
, 'sourceId' as k
, 2 as t
, TO_CHAR(prod_id) as v
, prod_id as vn FROM sh.products
ORDER BY 1,2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
432 rows affected.
```

Out[11]:

```
[]
```

3) Customers

Data by row

In [12]:

```
%%sql result <<
SELECT cust_id as vid
, 'label' as k
, 1 as t
, 'customer' as v
, null as vn FROM sh.customers
UNION ALL
SELECT cust_id as vid
, 'name' as k
, 1 as t
, cust_first_name || ' ' || cust_last_name as v
, null as vn FROM sh.customers
UNION ALL
SELECT cust_id as vid
, 'gender' as k
, 1 as t
, cust_gender as v
, null as vn FROM sh.customers
UNION ALL
SELECT cust_id as vid
, 'maritalStatus' as k
, 1 as t
, cust_marital_status as v
, null as vn FROM sh.customers
WHERE cust_marital_status IS NOT NULL
UNION ALL
SELECT cust_id as vid
, 'yearOfBirth' as k
, 2 as t
, TO_CHAR(cust_year_of_birth) as v
, cust_year_of_birth as vn FROM sh.customers
UNION ALL
SELECT cust_id as vid
, 'sourceId' as k
, 2 as t
, TO_CHAR(cust_id) as v
, cust_id as vn FROM sh.customers
ORDER BY 1,2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
Returning data to local variable result
```

In [13]:

```
result_df = result.DataFrame()
result_df.head(12)
```

Out[13]:

	vid	k	t	v	vn
0	1	gender	1	M	NaN
1	1	label	1	customer	NaN
2	1	name	1	Abigail Kessel	NaN
3	1	sourceId	2	1	1.0
4	1	yearOfBirth	2	1946	1946.0
5	2	gender	1	F	NaN
6	2	label	1	customer	NaN
7	2	name	1	Anne Koch	NaN
8	2	sourceId	2	2	2.0
9	2	yearOfBirth	2	1957	1957.0
10	3	gender	1	M	NaN
11	3	label	1	customer	NaN

Insert rows in MYSALESVT$

Because there isn't any date value, the 'vt' column isn't defined

In [14]:

```
%%sql
INSERT INTO MYSALESVT$ (vid, k, t, v, vn)
SELECT cust_id as vid
, 'label' as k
, 1 as t
, 'customer' as v
, null as vn FROM sh.customers
UNION ALL
SELECT cust_id as vid
, 'name' as k
, 1 as t
, cust_first_name || ' ' || cust_last_name as v
, null as vn FROM sh.customers
UNION ALL
SELECT cust_id as vid
, 'gender' as k
, 1 as t
, cust_gender as v
, null as vn FROM sh.customers
WHERE cust_gender IS NOT NULL
UNION ALL
SELECT cust_id as vid
, 'maritalStatus' as k
, 1 as t
, cust_marital_status as v
, null as vn FROM sh.customers
WHERE cust_marital_status IS NOT NULL
UNION ALL
SELECT cust_id as vid
, 'yearOfBirth' as k
, 2 as t
, TO_CHAR(cust_year_of_birth) as v
, cust_year_of_birth as vn FROM sh.customers
WHERE cust_year_of_birth IS NOT NULL
UNION ALL
SELECT cust_id as vid
, 'sourceId' as k
, 2 as t
, TO_CHAR(cust_id) as v
, cust_id as vn FROM sh.customers
ORDER BY 1,2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
315572 rows affected.
```

Out[14]:

```
[]
```

Quick check on the actual content of the graph (only nodes)

In [15]:

```
%%sql
SELECT v, COUNT(DISTINCT vid) FROM mysalesvt$
WHERE k = 'label'
GROUP BY v
ORDER BY 1
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[15]:

v	COUNT(DISTINCTVID)
country	23
customer	55500
product	72

In [16]:

```
%%sql
SELECT k, COUNT(DISTINCT vid) FROM mysalesvt$
GROUP BY k
ORDER BY 2 DESC,1
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[16]:

k	COUNT(DISTINCTVID)
label	55595
name	55595
sourceId	55595
gender	55500
yearOfBirth	55500
maritalStatus	38072
category	72
listPrice	72
subcategory	72
isoCode	23

Create edges (the orders)

1) Create a sequence

There isn't a real ID in the 'SALES' table and also there isn't one for the relation between customers and countries, therefore there isn't anything on which to build EID (edge ID)

In [17]:

```
%%sql
CREATE SEQUENCE mysales_eid_seq
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[17]:

```
[]
```

2) Customer -['livesIn']-> Country

In [18]:

```
%%sql result <<
SELECT null as eid
, cust_id as svid
, country_id as dvid
, 'livesIn' as el
, 'stateProvince' as k
, 1 as t
, cust_state_province as v FROM sh.customers
ORDER BY 2
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
Returning data to local variable result
```

In [19]:

```
result_df = result.DataFrame()
result_df.head()
```

Out[19]:

	eid	svid	dvid	el	k	t	v
0	None	1	52789	livesIn	stateProvince	1	England - Norfolk
1	None	2	52778	livesIn	stateProvince	1	Salamanca
2	None	3	52770	livesIn	stateProvince	1	Zeeland
3	None	4	52770	livesIn	stateProvince	1	Utrecht
4	None	5	52789	livesIn	stateProvince	1	England - Norfolk

Insert rows in MYSALESGE$

Because there isn't any date or numeric value, the 'vn' and 'vt' column aren't defined

In [20]:

```
%%sql
INSERT INTO MYSALESGE$ (eid, svid, dvid, el, k, t, v)
SELECT mysales_eid_seq.nextval
, svid, dvid, el, k, t, v FROM (
SELECT cust_id as svid
, country_id as dvid
, 'livesIn' as el
, 'stateProvince' as k
, 1 as t
, cust_state_province as v FROM sh.customers
ORDER BY 2
)
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
55500 rows affected.
```

Out[20]:

```
[]
```

3) Customer -['buys']-> Product

Create a temporary table to assign a unique ID acting as EID to sales using the sequence

In [21]:

```
%%sql
CREATE TABLE tmp_orders AS
SELECT mysales_eid_seq.nextval as eid
, svid, dvid, el, quantity_sold, amount_sold, order_date FROM (
SELECT null as eid
, cust_id as svid
, prod_id + 200000 as dvid
, 'buys' as el
, SUM(quantity_sold) as quantity_sold
, SUM(amount_sold) as amount_sold
, time_id as order_date FROM sh.sales
GROUP BY cust_id, prod_id, time_id
)
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[21]:

```
[]
```

In [22]:

```
%%sql result <<
SELECT eid
, svid
, dvid
, el
, 'quantity' as k
, 3 as t
, TO_CHAR(quantity_sold) as v
, quantity_sold as vn
, null as vt FROM tmp_orders
UNION ALL
SELECT eid
, svid
, dvid
, el
, 'amount' as k
, 3 as t
, TO_CHAR(amount_sold) as v
, amount_sold as vn
, null as vt FROM tmp_orders
UNION ALL
SELECT eid
, svid
, dvid
, el
, 'orderDate' as k
, 5 as t
, TO_CHAR(order_date, 'YYYY-MM-DD') as v
, null as vn
, order_date as vt FROM tmp_orders
ORDER BY 1,2,3,4,5
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
Returning data to local variable result
```

In [23]:

```
result_df = result.DataFrame()
result_df.head(12)
```

Out[23]:

	eid	svid	dvid	el	k	t	v	vn	vt
0	55501	2273	200013	buys	amount	3	1232.16	1232.16	NaT
1	55501	2273	200013	buys	orderDate	5	1998-01-10	None	1998-01-10
2	55501	2273	200013	buys	quantity	3	1	1	NaT
3	55502	1422	200013	buys	amount	3	1232.16	1232.16	NaT
4	55502	1422	200013	buys	orderDate	5	1998-01-20	None	1998-01-20
5	55502	1422	200013	buys	quantity	3	1	1	NaT
6	55503	3783	200013	buys	amount	3	1232.16	1232.16	NaT
7	55503	3783	200013	buys	orderDate	5	1998-01-20	None	1998-01-20
8	55503	3783	200013	buys	quantity	3	1	1	NaT
9	55504	6543	200013	buys	amount	3	1232.16	1232.16	NaT
10	55504	6543	200013	buys	orderDate	5	1998-01-20	None	1998-01-20
11	55504	6543	200013	buys	quantity	3	1	1	NaT

Insert rows in MYSALESGE$

In [24]:

```
%%sql
INSERT INTO MYSALESGE$ (eid, svid, dvid, el, k, t, v, vn, vt)
SELECT eid
, svid
, dvid
, el
, 'quantity' as k
, 3 as t
, TO_CHAR(quantity_sold) as v
, quantity_sold as vn
, null as vt FROM tmp_orders
UNION ALL
SELECT eid
, svid
, dvid
, el
, 'amount' as k
, 3 as t
, TO_CHAR(amount_sold) as v
, amount_sold as vn
, null as vt FROM tmp_orders
UNION ALL
SELECT eid
, svid
, dvid
, el
, 'orderDate' as k
, 5 as t
, TO_CHAR(order_date, 'YYYY-MM-DD') as v
, null as vn
, order_date as vt FROM tmp_orders
ORDER BY 1,2,3,4,5
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
2085399 rows affected.
```

Out[24]:

```
[]
```

Drop the temporary table and sequence (as it is just a one-shot loading)

In [25]:

```
%%sql
DROP TABLE tmp_orders
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[25]:

```
[]
```

In [26]:

```
%%sql
DROP SEQUENCE mysales_eid_seq
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[26]:

```
[]
```

Quick check on the actual content of the graph (only edges)

In [27]:

```
%%sql
SELECT el, COUNT(DISTINCT eid) FROM mysalesge$
GROUP BY el
ORDER BY 1
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[27]:

el	COUNT(DISTINCTEID)
buys	695133
livesIn	55500

In [28]:

```
%%sql
SELECT k, COUNT(DISTINCT eid) FROM mysalesge$
GROUP BY k
ORDER BY 2 DESC,1
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[28]:

k	COUNT(DISTINCTEID)
amount	695133
orderDate	695133
quantity	695133
stateProvince	55500

The graph content is now created into the database. In the next notebook (3%20Load%20graph%20from%20database.ipynb) the graph will be loaded into PGX and used for analysis.

Load a graph from database into PGX

The graph is stored in an Oracle 18c (18.3.0) database and will be loaded in the PGX server installed with the database.

In [1]:

```
%load_ext sql
%sql oracle://scott:Admin123@localhost:1521/?service_name=ORCLPDB1
```

Out[1]:

```
'Connected: scott@'
```

1) Prepare for loading

- setup environment (load libs etc.)
- connect to PGX
- build graph configuration

Import required packages

In [2]:

```
from jpype import *
import os
```

Setup & start JVM

In [3]:

```
# %load -s _get_pgx_class_path ../graphUtils.py
def _get_pgx_class_path(pgx_directory):
    class_path_list = []
    for root, dirs, files in os.walk(pgx_directory):
        for file in files:
            if file.endswith('.jar'):
                class_path_list.append(os.path.join(root, file))
    return ':'.join(class_path_list)
```

In [4]:

```
pgxLibsPath = '/opt/oracle/product/18c/dbhome_1/md/property_graph/lib'
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path(pgxLibsPa
th))
```

Create session

Need to start the PGX 3.1.1 server before to continue:

```
$ORACLE_HOME/md/property_graph/pgx/bin/start-server
```

In [5]:

```
session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7007/", "my_sess
ion")
print(session)
```

```
PgxSession[ID=a595284d-8182-485a-b37b-677074c76e9c,source=my_session]
```

Build config for nodes and edges properties (full load of all the possible existing properties)

In [6]:

```
%%sql
WITH properties AS (
    SELECT DISTINCT k, t, 'Vertex' AS kind
    FROM mysalesvt$
    UNION ALL
    SELECT DISTINCT k, t, 'Edge' AS kind
    FROM mysalesge$
)
,cfg AS (
    SELECT '.add' || kind || 'Property("' || k || '",PropertyTypeClass.'
         || CASE WHEN t = 1 THEN 'STRING' WHEN t = 2 THEN 'INTEGER' WHEN t = 3 THEN 'FL
OAT' WHEN t = 5 THEN 'DATE' WHEN t = 6 THEN 'BOOLEAN' END
         || ')' AS prop
    FROM properties
) SELECT LISTAGG(prop,'') WITHIN GROUP(ORDER BY prop) FROM cfg
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[6]:

.addEdgeProperty("amount",PropertyTypeClass.FLOAT).addEdgeProperty("orderDate",PropertyTypeCl

Need to replace 'DATE' types

`date` types are still not supported in PGQL as results or sort elements, therefore it's easier to "cheat" and load the date as if it would be a `string`

In [7]:

```
GraphConfigBuilderClass = JClass('oracle.pgx.config.GraphConfigBuilder')
PropertyTypeClass = JClass('oracle.pgx.common.types.PropertyType')

cfg = GraphConfigBuilderClass.forPropertyGraphRdbms()\
    .setUsername("scott")\
    .setPassword("Admin123")\
    .setName("mysales")\
    .setMaxNumConnections(4)\
    .setJdbcUrl("jdbc:oracle:thin:@localhost:1521/ORCLPDB1")\
    .setLoadEdgeLabel(True)\
    .setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSSSSXXX")\
    .addEdgeProperty("amount",PropertyTypeClass.FLOAT).addEdgeProperty("orderDate",Prop
ertyTypeClass.STRING).addEdgeProperty("quantity",PropertyTypeClass.FLOAT).addEdgeProper
ty("stateProvince",PropertyTypeClass.STRING).addVertexProperty("category",PropertyTypeC
lass.STRING).addVertexProperty("gender",PropertyTypeClass.STRING).addVertexProperty("is
oCode",PropertyTypeClass.STRING).addVertexProperty("label",PropertyTypeClass.STRING).ad
dVertexProperty("listPrice",PropertyTypeClass.FLOAT).addVertexProperty("maritalStatus",
PropertyTypeClass.STRING).addVertexProperty("name",PropertyTypeClass.STRING).addVertexP
roperty("sourceId",PropertyTypeClass.INTEGER).addVertexProperty("subcategory",PropertyT
ypeClass.STRING).addVertexProperty("yearOfBirth",PropertyTypeClass.INTEGER)
cfg = cfg.build()

print(cfg)
```


```
{"error_handling":{},"date_format":"yyyy-MM-dd'T'HH:mm:ss.SSSSSSXXX","vert
ex_id_type":"long","max_num_connections":4,"format":"pg","attributes":
{},"password":"*******","db_engine":"RDBMS","vertex_props":[{"name":"categ
ory","type":"string"},{"name":"gender","type":"string"},{"name":"isoCod
e","type":"string"},{"name":"label","type":"string"},{"name":"listPric
e","type":"float"},{"name":"maritalStatus","type":"string"},{"name":"nam
e","type":"string"},{"name":"sourceId","type":"integer"},{"name":"subcateg
ory","type":"string"},{"name":"yearOfBirth","type":"integer"}],"name":"mys
ales","loading":{"load_edge_label":true},"edge_props":[{"name":"amount","t
ype":"float"},{"name":"orderDate","type":"string"},{"name":"quantity","typ
e":"float"},{"name":"stateProvince","type":"string"}],"jdbc_url":"jdbc:ora
cle:thin:@localhost:1521/ORCLPDB1","username":"scott"}
```


3) Load graph

In [8]:

```
OraclePropertyGraphClass = JClass('oracle.pg.rdbms.OraclePropertyGraph')
opg = OraclePropertyGraphClass.getInstance(cfg)

print(opg)
```

```
oraclepropertygraph with name mysales
```

In [9]:

```
pgxGraph = session.readGraphWithProperties(opg.getConfig())

print(pgxGraph)
```

```
PgxGraph[name=mysales_2,N=55595,E=695133,created=1551460273281]
```

Test graph

Count number of nodes and edges

In [10]:

```
print('Graph has ' + str(pgxGraph.getNumEdges())    + ' edges')
print('Graph has ' + str(pgxGraph.getNumVertices()) + ' vertices')
```

```
Graph has 695133 edges
Graph has 55595 vertices
```

4) Use the graph

Sample PGQL query

PGQL specification can be found at http://pgql-lang.org/ (http://pgql-lang.org/)
PGX 2.5.1 coming with Oracle Database 18c supports PGQL 1.0, PGX 2.6.1+ supports PGQL 1.1

In [11]:

```
query = ("SELECT c.name, p.name, b.orderDate, b.amount, b.quantity "
         "WHERE (c WITH label = 'customer') -[b:buys]-> (p WITH label = 'product') "
         "ORDER BY b.orderDate, c.name LIMIT 10"
        )
pgxResultSet = pgxGraph.queryPgql(query)

print(pgxResultSet)
print('----------------')

pgxResults = pgxResultSet.getResults()
for r in pgxResults.iterator():
    print("{} bought a quantity of {} of '{}' for a price of {} on {}".format(r.getStri
ng(0), r.getFloat(4), r.getString(1), r.getFloat(3), r.getString(2)[:10]))
```

```
PgqlResultSetImpl[graph=mysales_2,numResults=10]
----------------
Abigail Ruddy bought a quantity of 1.0 of 'Envoy External 8X CD-ROM' for a
price of 63.57 on 1998-01-01
Adel Peebles bought a quantity of 2.0 of 'Deluxe Mouse' for a price of 61.
01 on 1998-01-01
Amarylis Nenninger bought a quantity of 1.0 of 'O/S Documentation Set - En
glish' for a price of 47.69 on 1998-01-01
Anand Rowley bought a quantity of 1.0 of 'Envoy External 8X CD-ROM' for a
price of 63.57 on 1998-01-01
Anand Rowley bought a quantity of 1.0 of '3 1/2" Bulk diskettes, Box of 10
0' for a price of 30.15 on 1998-01-01
Anand Rowley bought a quantity of 1.0 of '3 1/2" Bulk diskettes, Box of 5
0' for a price of 16.63 on 1998-01-01
Anand Rowley bought a quantity of 1.0 of 'Internal 8X CD-ROM' for a price
of 40.45 on 1998-01-01
Bailey Thompson bought a quantity of 1.0 of 'O/S Documentation Set - Engli
sh' for a price of 47.69 on 1998-01-01
Baird Rogers bought a quantity of 2.0 of 'CD-RW, High Speed, Pack of 10' f
or a price of 18.6 on 1998-01-01
Baird Rogers bought a quantity of 1.0 of 'Music CD-R' for a price of 19.64
on 1998-01-01
```

In [12]:

```
# %load -s pgql2dictionary ../graphUtils.py
def pgql2dictionary(pgxResultSet):
    dk = {}
    resultElements = pgxResultSet.getPgqlResultElements()
    for i in range(len(resultElements)):
        re = resultElements.get(i)
        dk[re.getVarName()] = str(re.getElementType())

    # define the dictionary
    d = {}
    # add the empty list to dictionary
    for k in dk:
        d[k] = []

    # append values
    pgxResults = pgxResultSet.getResults()
    for r in pgxResults.iterator():
        for k in dk:
            if dk[k] == 'string':
                d[k].append(r.getString(k))
            elif dk[k] == 'vertex':
                d[k].append('vertex('+str(r.getVertex(k).getId())+')')
            elif dk[k] == 'edge':
                d[k].append('edge('+str(r.getEdge(k).getId())+')')
            elif dk[k] == 'long':
                d[k].append(r.getLong(k))
            elif dk[k] == 'double':
                d[k].append(r.getDouble(k))
            elif dk[k] == 'float':
                d[k].append(r.getFloat(k))
            else:
                #print(dk[k])
                d[k].append('N/A')
    return d
```

In [13]:

```
import pandas as pd

query = ("SELECT p.name as prod_name, SUM(b.quantity) as total_quantity, SUM(b.amount)
 as total_amount "
         "WHERE (p WITH label = 'product') <-[b:buys]- (c) "
         "GROUP BY p.name"
        )
pgxResultSet = pgxGraph.queryPgql(query)
print(pgxResultSet)

df = pd.DataFrame(pgql2dictionary(pgxResultSet))
df.head()
```

```
PgqlResultSetImpl[graph=mysales_2,numResults=71]
```

Out[13]:

	prod_name	total_quantity	total_amount
0	External 6X CD-ROM	13043.0	577580.352684
1	O/S Documentation Set - German	12429.0	604081.908741
2	Comic Book Heroes	4572.0	101214.599781
3	Deluxe Mouse	12837.0	377400.310974
4	Music CD-R	14315.0	301848.198940

In [14]:

```
df.describe(include='all')
```

Out[14]:

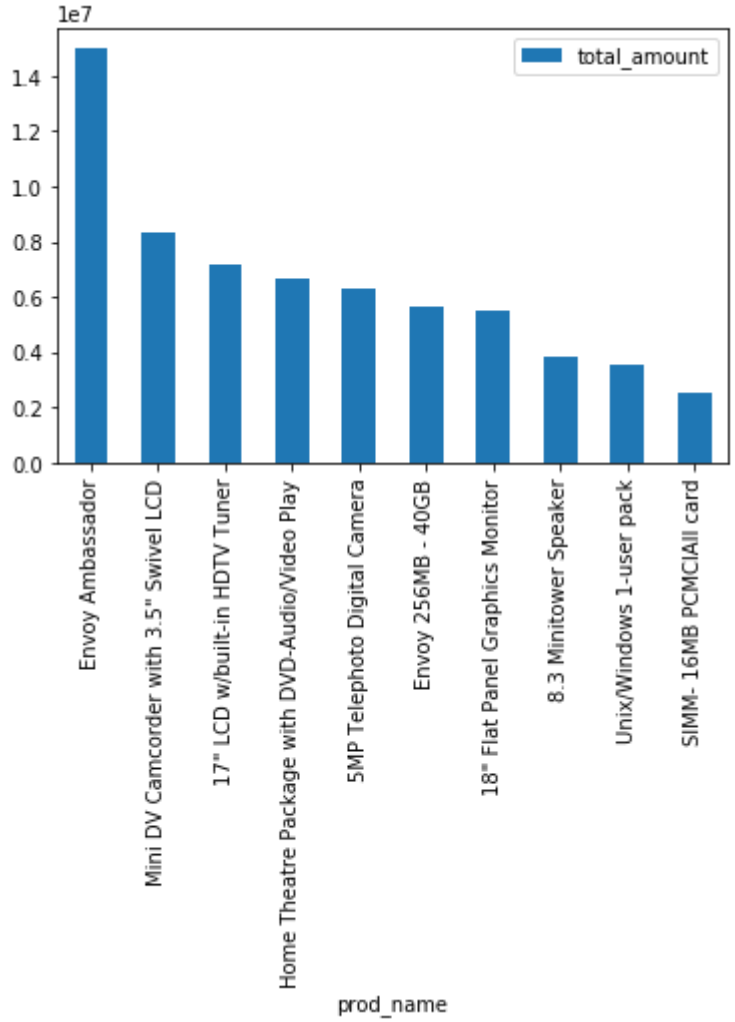
	prod_name	total_quantity	total_amount
count	71	71.000000	7.100000e+01
unique	71	NaN	NaN
top	O/S Documentation Set - Spanish	NaN	NaN
freq	1	NaN	NaN
mean	NaN	12941.450704	1.383181e+06
std	NaN	6024.180200	2.461307e+06
min	NaN	710.000000	2.793333e+04
25%	NaN	8092.000000	2.572828e+05
50%	NaN	12429.000000	5.130911e+05
75%	NaN	16613.000000	1.040238e+06
max	NaN	29282.000000	1.501164e+07

In [15]:

```
%matplotlib inline
df.nlargest(10, 'total_amount').plot(kind='bar',x='prod_name',y='total_amount')
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f233c01ca20>
```

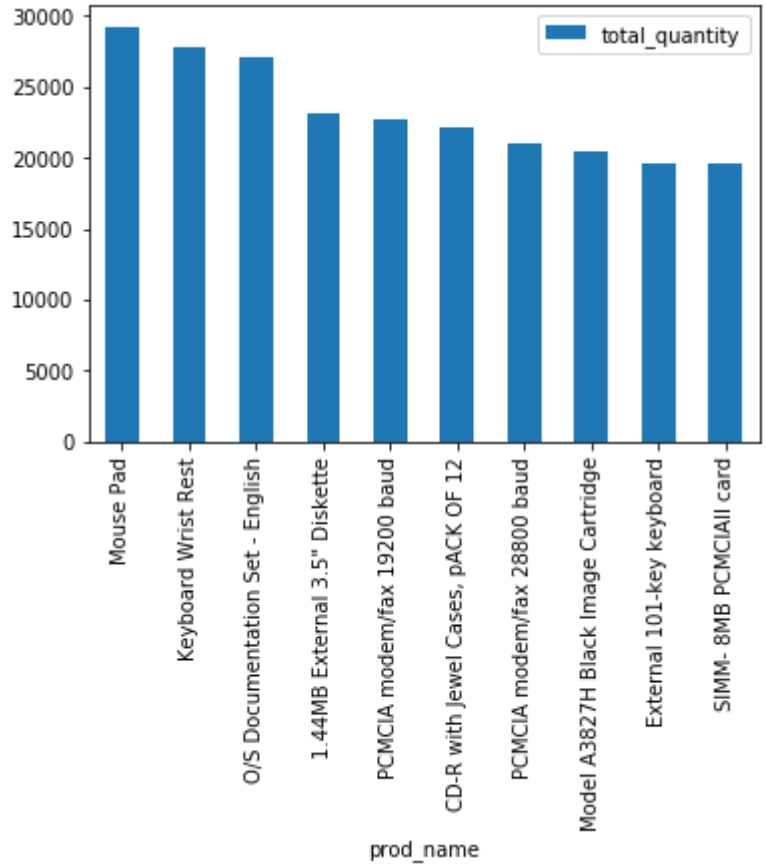


In [16]:

```
df.nlargest(10, 'total_quantity').plot(kind='bar',x='prod_name',y='total_quantity')
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22e42061d0>
```



Analysis with graph algorithms

Run a pagerank algorithm on the graph to find the most important (top pagerank score) nodes.
A reasonable guess is the result will have products of the *TOP 10 by quantity* above as they probably are the nodes with more connections.

In [17]:

```
analyst = session.createAnalyst()
pagerank = analyst.pagerank(pgxGraph)
query = ("SELECT x, x.label, x.name, x."+pagerank.getName()+" "
         "WHERE (x) ORDER BY x."+pagerank.getName()+" DESC LIMIT 10"
        )

pgxResultSet = pgxGraph.queryPgql(query)

print(pgxResultSet)
print('--------------------')

pgxResults = pgxResultSet.getResults()
for r in pgxResults.iterator():
    print("{}: {} - page rank = {}".format(r.getString(1), r.getString(2), r.getDouble(
3)))
```

```
PgqlResultSetImpl[graph=mysales_2,numResults=10]
--------------------
product: Mouse Pad - page rank = 7.002237185165857E-4
product: Keyboard Wrist Rest - page rank = 6.045444156492589E-4
product: O/S Documentation Set - English - page rank = 5.955638280537223E-
4
product: External 8X CD-ROM - page rank = 4.8124497979426615E-4
product: SIMM- 16MB PCMCIAII card - page rank = 4.6121060880009084E-4
product: CD-RW, High Speed Pack of 5 - page rank = 4.5177898111075276E-4
product: Model SM26273 Black Ink Cartridge - page rank = 4.461510617252293
6E-4
product: PCMCIA modem/fax 19200 baud - page rank = 3.75975313849342E-4
product: 1.44MB External 3.5" Diskette - page rank = 3.741326066667993E-4
product: Standard Mouse - page rank = 3.7212800945481855E-4
```

Cleanup (free memory now)

In [18]:

```
pgxGraph.destroy()
```

Drop the graph in the database

Call a package method to drop the graph (and the related tables)
This is equivalent to

```
BEGIN
    OPG_APIS.DROP_PG('mysales');
END;
```

In [19]:

```
import cx_Oracle
con = cx_Oracle.connect('scott/Admin123@localhost:1521/ORCLPDB1')
cur = con.cursor()
cur.callproc('OPG_APIS.DROP_PG', ['mysales'])
cur.close()
con.close()
```

Save and load graphs from files

A graph can be saved and loaded as a file (or various files) on disk (mainly useful when not storing in a database). Various formats are supported including a binary specific one which provides the most feature (nodes labels, various nodes IDs type etc.).

Import required packages

In [1]:

```
from jpype import *
import os
```

Setup & start JVM

In [2]:

```
# %load -s _get_pgx_class_path ../graphUtils.py
def _get_pgx_class_path(pgx_directory):
    class_path_list = []
    for root, dirs, files in os.walk(pgx_directory):
        for file in files:
            if file.endswith('.jar'):
                class_path_list.append(os.path.join(root, file))
    return ':'.join(class_path_list)
```

In [3]:

```
pgxLibsPath = '/opt/oracle/product/18c/dbhome_1/md/property_graph/lib'
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path(pgxLibsPa
th))
```

Create a session on the PGX 3.1.1 server

Need to start the PGX 3.1.1 server before to continue:

```
$ORACLE_HOME/md/property_graph/pgx/bin/start-server
```

In [4]:

```
session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7007/", "my_sess
ion")
```

Create a new graph

In [5]:

```
# new builder
builder = session.newGraphBuilder(JClass('oracle.pgx.common.types.IdType').LONG)

# define some nodes (node ID is unique!)
builder.addVertex(1).addLabel("person").setProperty("name", "Francesco").setProperty("c
ountry", "Italy")
builder.addVertex(2).addLabel("person").setProperty("name", "Christian").setProperty("c
ountry", "Switzerland")
builder.addVertex(3).addLabel("session").setProperty("name", "Starting an Oracle Analyt
ics Cloud Journey from 0")
builder.addVertex(4).addLabel("event").setProperty("name", "ITOUG 2019")
builder.addVertex(5).addLabel("country").setProperty("name", "Italy")
builder.addVertex(6).addLabel("country").setProperty("name", "Switzerland")
builder.addVertex(7).addLabel("continent").setProperty("name", "Europe")

# define some edges (edge ID is unique!)
builder.addEdge(0, 1, 2).setLabel("friendOf")
builder.addEdge(1, 1, 3).setLabel("presents")
builder.addEdge(2, 2, 3).setLabel("presents")
builder.addEdge(3, 3, 4).setLabel("scheduledAt")
builder.addEdge(4, 1, 5).setLabel("livesIn")
builder.addEdge(5, 2, 6).setLabel("livesIn")
builder.addEdge(6, 5, 7).setLabel("partOf")
builder.addEdge(7, 6, 7).setLabel("partOf")
builder.addEdge(8, 4, 5).setLabel("userGroupOf")
```

Out[5]:

```
<jpype._jclass.oracle.pgx.api.graphbuilder.EdgeBuilderImpl at 0x7f0534478c
88>
```

Build new graph

In [6]:

```
graph = builder.build()

print(graph)
```

```
PgxGraph[name=anonymous_graph_15,N=7,E=9,created=1551460444407]
```

Visualize the graph

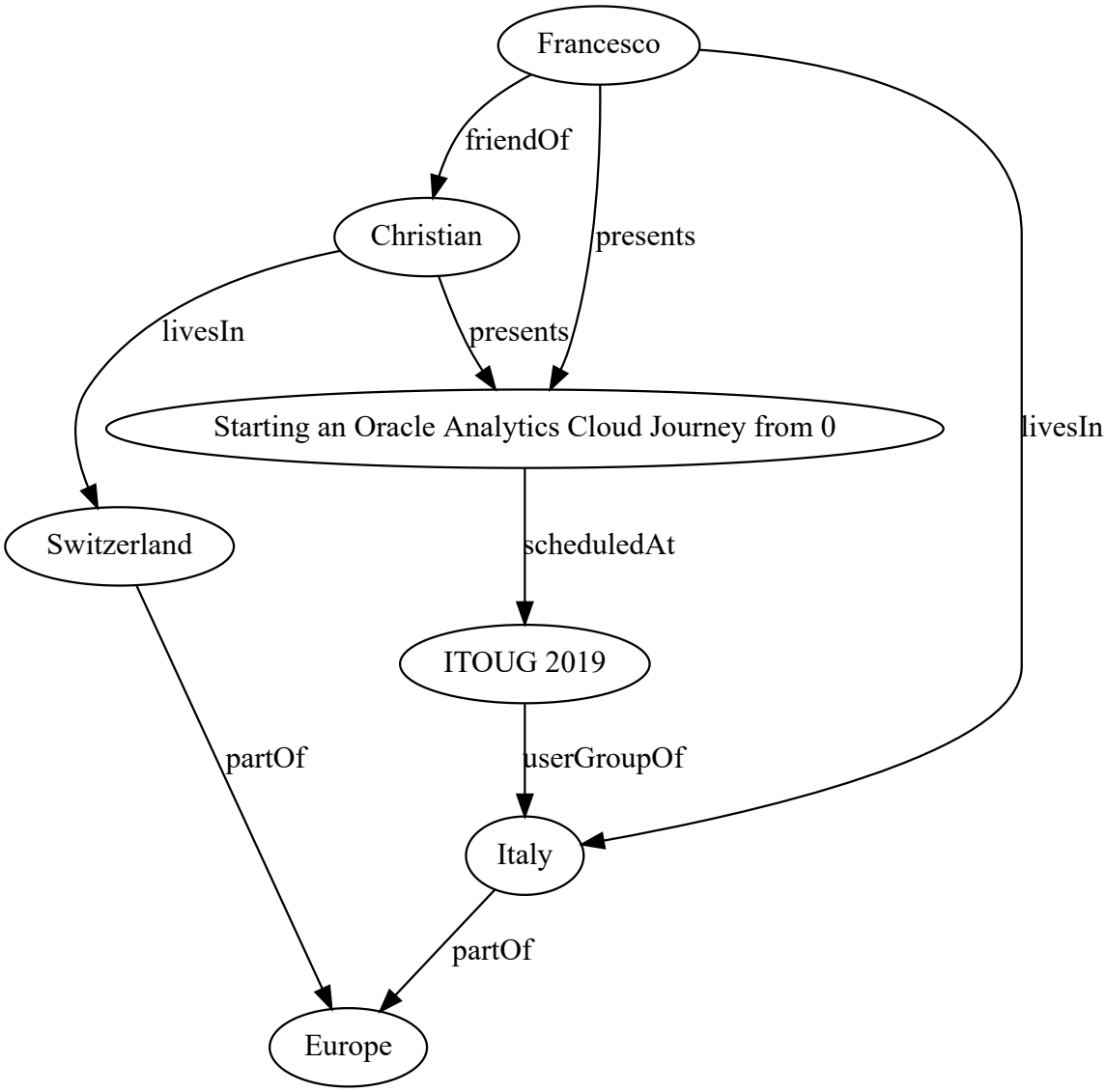
In [7]:

```
# %load -s renderGraph ../graphUtils.py
def renderGraph(graph):
    from graphviz import Digraph

    # get all the vertices of the graph
    vertices = graph.getVertices()
    # create a new visualization
    dot = Digraph(comment='Graph')
    # loop over vertices
    for v in vertices.iterator():
        dot.node(str(v.getId()), v.getProperty("name"))

    # loop over vertices to get 'out' edges
    for v in vertices.iterator():
        edges = v.getOutEdges()
        # loop over 'out' edges
        for e in edges:
            dot.edge(str(e.getSource().getId()), str(e.getDestination().getId()), label
=e.getLabel())

    # return (display) graph
    return dot
```

In [8]:

```
renderGraph(graph)
```

Out[8]:


Francesco
friendOf
Christian
presents
livesIn
presents
Starting an Oracle Analytics Cloud Journey from 0
livesIn
Switzerland
scheduledAt
ITOUG 2019
partOf
userGroupOf
Italy
partOf
Europe

Save file on disk

In [9]:

```
currDirectory = !pwd
storeIn = currDirectory[0]+'/sample_graph.pgb'

storeConfig = graph.store(JClass('oracle.pgx.config.Format').PGB, storeIn)
cfgStoreFile = open(storeIn+'.json', 'w')
cfgStoreFile.write(storeConfig.toString())
cfgStoreFile.close()

print(storeConfig)
```

{"loading":{"load_edge_label":true,"load_vertex_labels":true},"error_handl
ing":{},"edge_props":[],"vertex_uris":["/opt/jupyter/git/jupyterlab/Graph
database from scratch/sample_graph.pgb"],"edge_uris":[],"format":"pgb","ve
rtex_id_type":"long","attributes":{},"vertex_props":[{"type":"string","nam
e":"name"},{"type":"string","name":"country"}]}

Load graph from disk

In [10]:

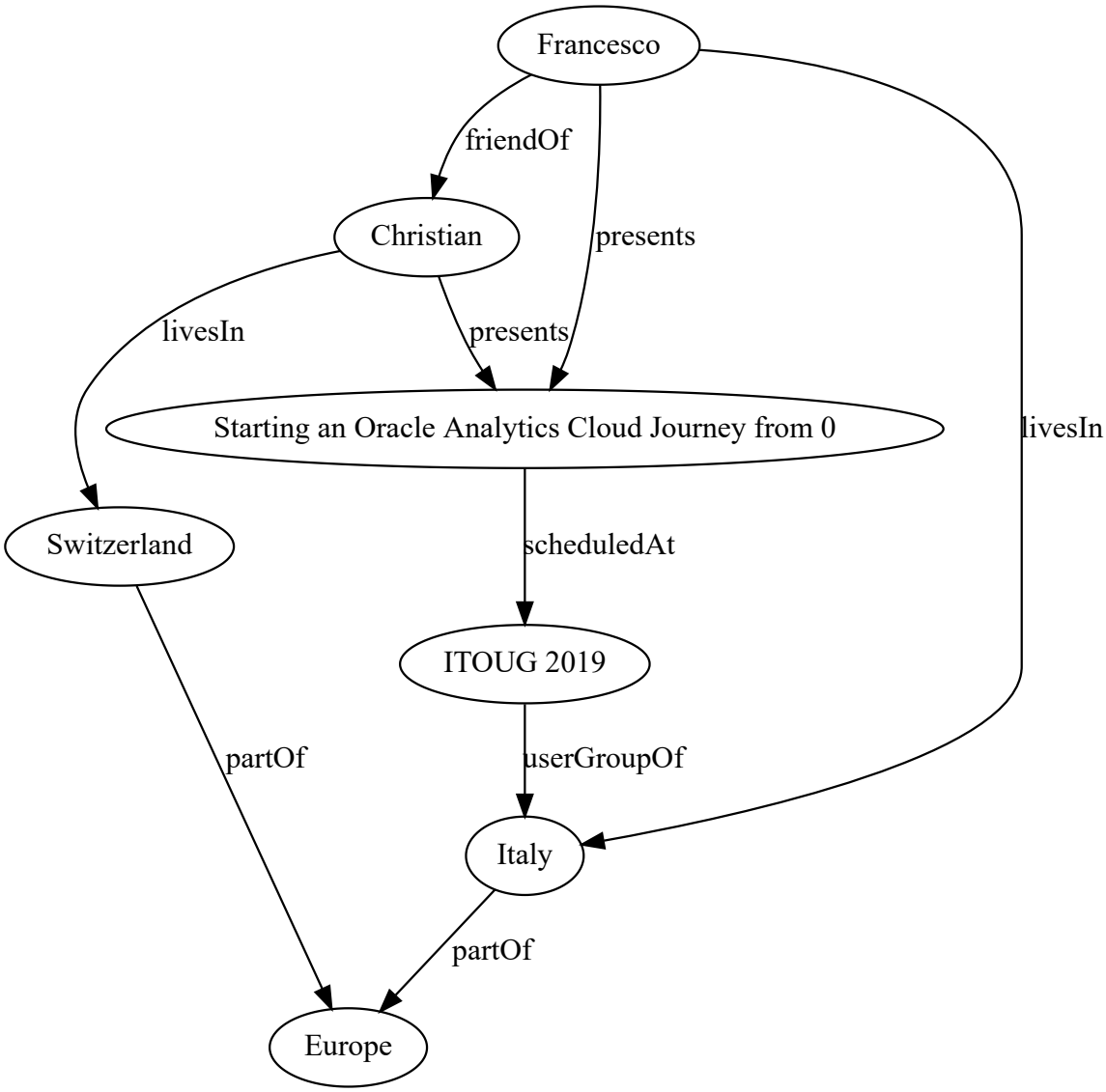
```
graphConfigFile = storeIn+'.json'

myGraph = session.readGraphWithProperties(graphConfigFile)

print(myGraph)
```

PgxGraph[name=sample_graph,N=7,E=9,created=1551460463683]

In [11]:

```
renderGraph(myGraph)
```

Out[11]:



Cleanup

Delete files on disk (the graph file and the graph config JSON file)

In [12]:

```
%%bash
rm sample_graph.pgb
rm sample_graph.pgb.json
```

Convert PGQL to SQL

When using an Oracle Database as storage it is possible to translate a PGQL query on graph in PGX into a SQL query to be executed directly in the database (on the graph tables *graphVT$* and *graphGE$*).

Import required packages

In [1]:

```
from jpype import *
import os
```

Setup JVM

Start JVM passing the PGX 3.1.1 classpath.
getDefaultJVMPath() will try to detect the location of java in the system, it can be replaced by the string defining the path to `libjvm.so` .

In [2]:

```
# %load -s _get_pgx_class_path ../graphUtils.py
def _get_pgx_class_path(pgx_directory):
    class_path_list = []
    for root, dirs, files in os.walk(pgx_directory):
        for file in files:
            if file.endswith('.jar'):
                class_path_list.append(os.path.join(root, file))
    return ':'.join(class_path_list)
```

In [3]:

```
pgxLibsPath = '/opt/oracle/product/18c/dbhome_1/md/property_graph/lib'
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path(pgxLibsPa
th))
```

Convert PGQL to SQL

Based on the documentation (https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/sql-based-property-graph-query-analytics.html#GUID-7642327B-B973-4C48-90B1-1447F3D57CA5 (https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/sql-based-property-graph-query-analytics.html#GUID-7642327B-B973-4C48-90B1-1447F3D57CA5)) it's possible to translate PGQL into SQL without executing it.

Sample PGQL query

In [4]:

```
pgql = ("SELECT c.name, p.name, b.orderDate, b.amount, b.quantity "
        "WHERE (c WITH label = 'customer') -[b:buys]-> (p WITH label = 'product') LIMIT
10"
       )
```

Convert into SQL

If the graph defined doesn't exist and empty one is created in the database.

Some parameters to "drive" the generated SQL exists, details in https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/sql-based-property-graph-query-analytics.html#GUID-E9CC82C3-BD5A-4581-AE26-2432D6929D44 (https://docs.oracle.com/en/database/oracle/oracle-database/18/spgdg/sql-based-property-graph-query-analytics.html#GUID-E9CC82C3-BD5A-4581-AE26-2432D6929D44). For example the usage of the GT$ table.

In [5]:

```
# Define Java>Python classes
OracleClass = JClass('oracle.pg.rdbms.Oracle')
OraclePropertyGraphClass = JClass('oracle.pg.rdbms.OraclePropertyGraph')
OraclePgqlExecutionFactoryClass = JClass('oracle.pg.rdbms.OraclePgqlExecutionFactory')

# Create a connection to Oracle
oracle = OracleClass('jdbc:oracle:thin:@localhost:1521/ORCLPDB1', 'scott', 'Admin123')
# Select property graph
opg = OraclePropertyGraphClass.getInstance(oracle, 'mysales')

# Create an OraclePgqlStatement
ops = OraclePgqlExecutionFactoryClass.createStatement(opg)

# Get the SQL translation (the 'pgql' query was defined in the previous cell)
sqlTrans = ops.translateQuery(pgql, "")

print(pgql)
print('----------------------------------------')
print(sqlTrans.getSqlTranslation())
```

```
SELECT c.name, p.name, b.orderDate, b.amount, b.quantity WHERE (c WITH lab
el = 'customer') -[b:buys]-> (p WITH label = 'product') LIMIT 10
----------------------------------------
SELECT * FROM(SELECT T0$4.T AS "c.name$T",
T0$4.V AS "c.name$V",
T0$4.VN AS "c.name$VN",
T0$4.VT AS "c.name$VT",
T0$1.T AS "p.name$T",
T0$1.V AS "p.name$V",
T0$1.VN AS "p.name$VN",
T0$1.VT AS "p.name$VT",
T0$3.T AS "b.orderDate$T",
T0$3.V AS "b.orderDate$V",
T0$3.VN AS "b.orderDate$VN",
T0$3.VT AS "b.orderDate$VT",
T0$0.T AS "b.amount$T",
T0$0.V AS "b.amount$V",
T0$0.VN AS "b.amount$VN",
T0$0.VT AS "b.amount$VT",
T0$2.T AS "b.quantity$T",
T0$2.V AS "b.quantity$V",
T0$2.VN AS "b.quantity$VN",
T0$2.VT AS "b.quantity$VT"
FROM "SCOTT".MYSALESGE$ T0$0,
"SCOTT".MYSALESVT$ T0$1,
"SCOTT".MYSALESGE$ T0$2,
"SCOTT".MYSALESGE$ T0$3,
"SCOTT".MYSALESVT$ T0$4
WHERE T0$0.K=n'amount' AND
T0$1.K=n'name' AND
T0$2.K=n'quantity' AND
T0$3.K=n'orderDate' AND
T0$4.K=n'name' AND
T0$0.DVID=T0$1.VID AND
T0$0.EID=T0$2.EID AND
T0$0.EID=T0$3.EID AND
T0$0.SVID=T0$4.VID AND
(T0$4.T = 1 AND T0$4.V = n'customer') AND
(T0$1.T = 1 AND T0$1.V = n'product') AND
(T0$0.EL = n'buys' AND T0$0.EL IS NOT NULL))
WHERE ROWNUM <= 10
```

Test SQL

Execute the generated query on the database

In [6]:

```
%load_ext sql
%sql oracle://scott:Admin123@localhost:1521/?service_name=ORCLPDB1
```

Out[6]:

```
'Connected: scott@'
```

In [7]:

```
%%sql
SELECT * FROM(SELECT T0$4.T AS "c.name$T",
T0$4.V AS "c.name$V",
T0$4.VN AS "c.name$VN",
T0$4.VT AS "c.name$VT",
T0$1.T AS "p.name$T",
T0$1.V AS "p.name$V",
T0$1.VN AS "p.name$VN",
T0$1.VT AS "p.name$VT",
T0$3.T AS "b.orderDate$T",
T0$3.V AS "b.orderDate$V",
T0$3.VN AS "b.orderDate$VN",
T0$3.VT AS "b.orderDate$VT",
T0$0.T AS "b.amount$T",
T0$0.V AS "b.amount$V",
T0$0.VN AS "b.amount$VN",
T0$0.VT AS "b.amount$VT",
T0$2.T AS "b.quantity$T",
T0$2.V AS "b.quantity$V",
T0$2.VN AS "b.quantity$VN",
T0$2.VT AS "b.quantity$VT"
FROM "SCOTT".MYSALESGE$ T0$0,
"SCOTT".MYSALESVT$ T0$1,
"SCOTT".MYSALESGE$ T0$2,
"SCOTT".MYSALESGE$ T0$3,
"SCOTT".MYSALESVT$ T0$4
WHERE T0$0.K=n'amount' AND
T0$1.K=n'name' AND
T0$2.K=n'quantity' AND
T0$3.K=n'orderDate' AND
T0$4.K=n'name' AND
T0$0.DVID=T0$1.VID AND
T0$0.EID=T0$2.EID AND
T0$0.EID=T0$3.EID AND
T0$0.SVID=T0$4.VID AND
(T0$4.T = 1 AND T0$4.V = n'customer') AND
(T0$1.T = 1 AND T0$1.V = n'product') AND
(T0$0.EL = n'buys' AND T0$0.EL IS NOT NULL))
WHERE ROWNUM <= 10
```

```
 * oracle://scott:***@localhost:1521/?service_name=ORCLPDB1
0 rows affected.
```

Out[7]:

c.name$T	c.name$V	c.name$VN	c.name$VT	p.name$T	p.name$V	p.name$VN	p.name$VT