ITOUG
ITALIAN ORACLE USER GROUP

Graph Database

From scratch in 45 minutes

Gianni Ceresa

Managing Director of DATAlysis GmbH (Switzerland)

Working with BI and EPM tools for about 11 years

Oracle ACE

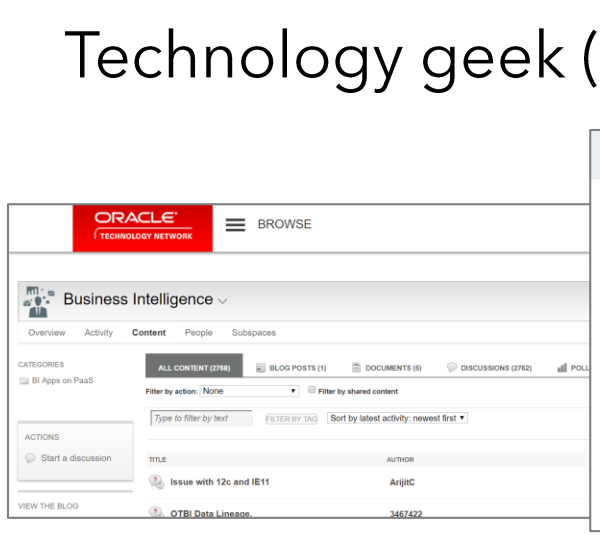
Part-time blogger on **gianniceresa.com**

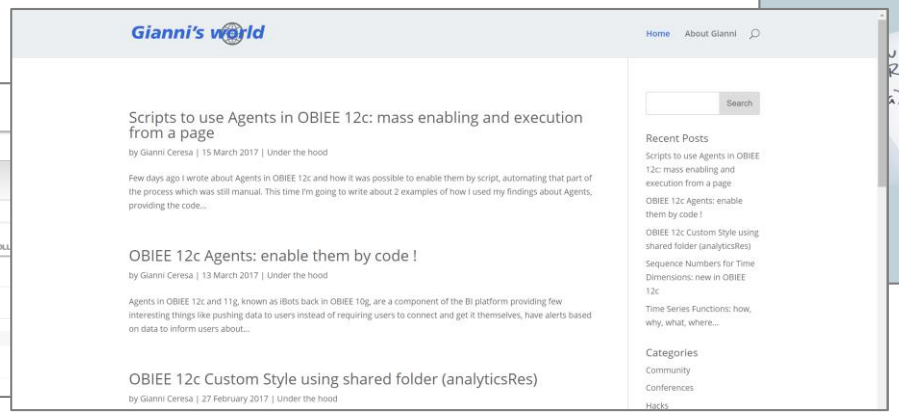
Full-time IRC (freenode | #obihackers) resident

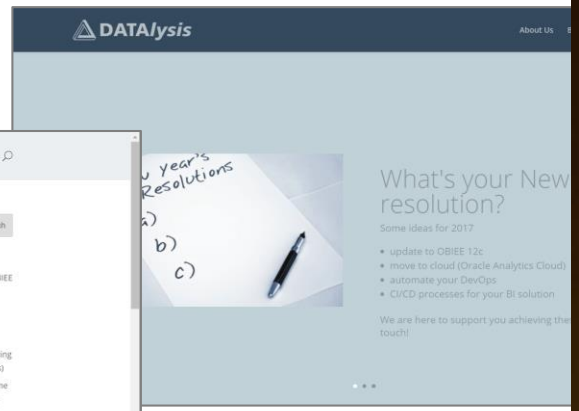
Same group on Telegram **http://telegram.me/obihackers**

ODC (ex OTN) forums addict

Technology geek (or just geek in general)



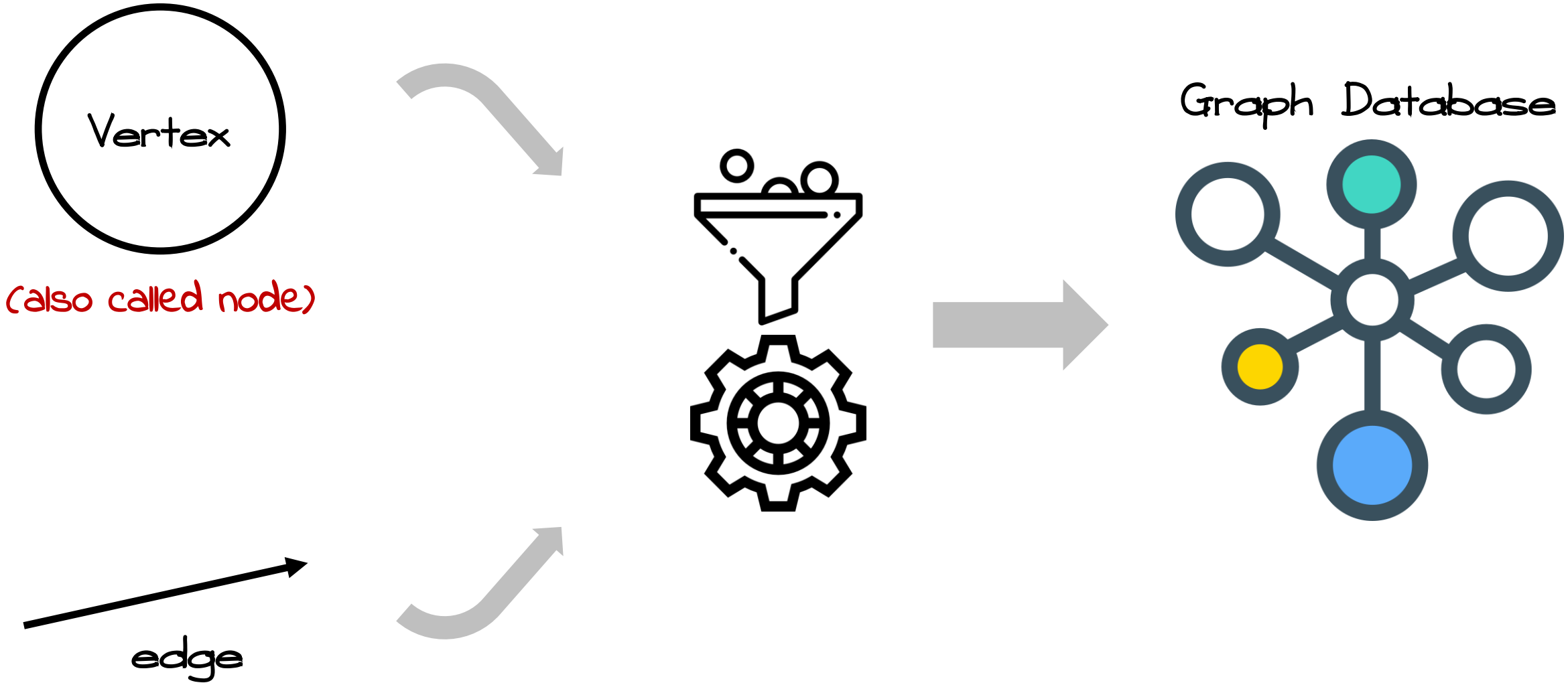




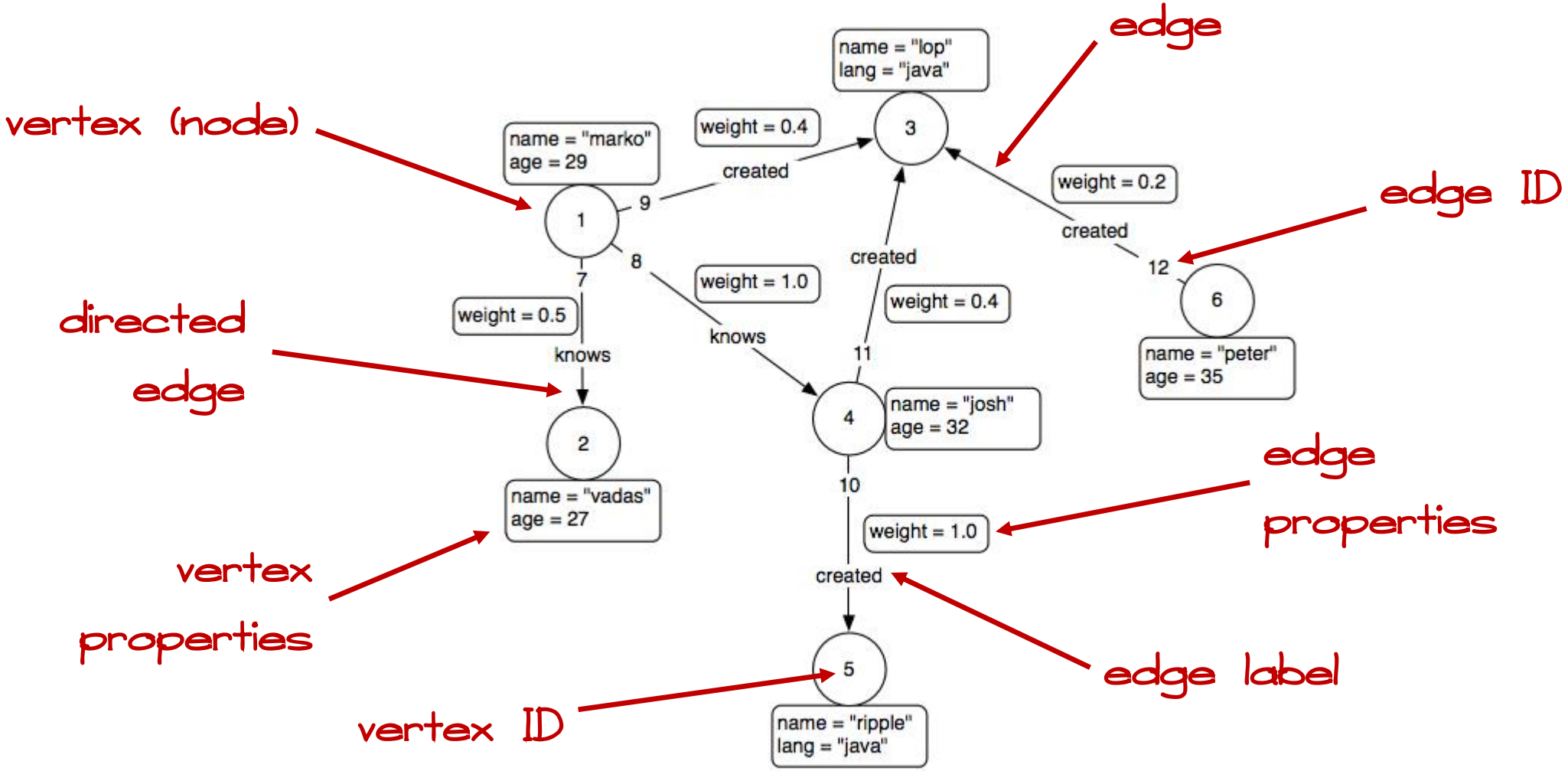
@G_Ceresa

DATAlysis

Graph Database – What's that?



Graph Database – What's that?



Where to start?

Get Oracle Labs Parallel Graph AnalytiX (PGX)

- From OTN: https://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytix/downloads/index.html
- From your Big Data Appliance
- From your Oracle Database (>= 12c R2)

The OTN version is just a ZIP file

- Linux or Mac (no Windows for now)
- Requires JDK8

Didn't test it fully, got some of it working on windows embedded in a Java project, and a .bat file is included in the ZIP

Useful resources

- Tutorial: https://docs.oracle.com/cd/E56133_01/latest/tutorials/index.html
- Reference guides: https://docs.oracle.com/cd/E56133_01/latest/reference/index.html
- Javadocs: https://docs.oracle.com/cd/E56133_01/latest/javadocs/index.html

Where to start?

This presentation and the examples and code used is based on

- PGX 2.5.1 (when using PGX embedded in Oracle Database 18c)
- PGX 2.7.0 (when using the standalone version)

Version 3.2.0 has been released last Monday (the site says 21.1, the links and doc appeared online only the 28.1).

It brings some major changes, new feature, remove some old things and also break retro-compatibility with older clients because of changes in the API.

The new 3.2.0 Javadocs doesn't contain many basic things anymore, therefore the tutorials are either still based on old versions code or the new Javadocs missed many methods (which would make it useless).

Build a graph by hand

Apache Zeppelin has an interpreter for PGX which allows to easily connect and use PGX.

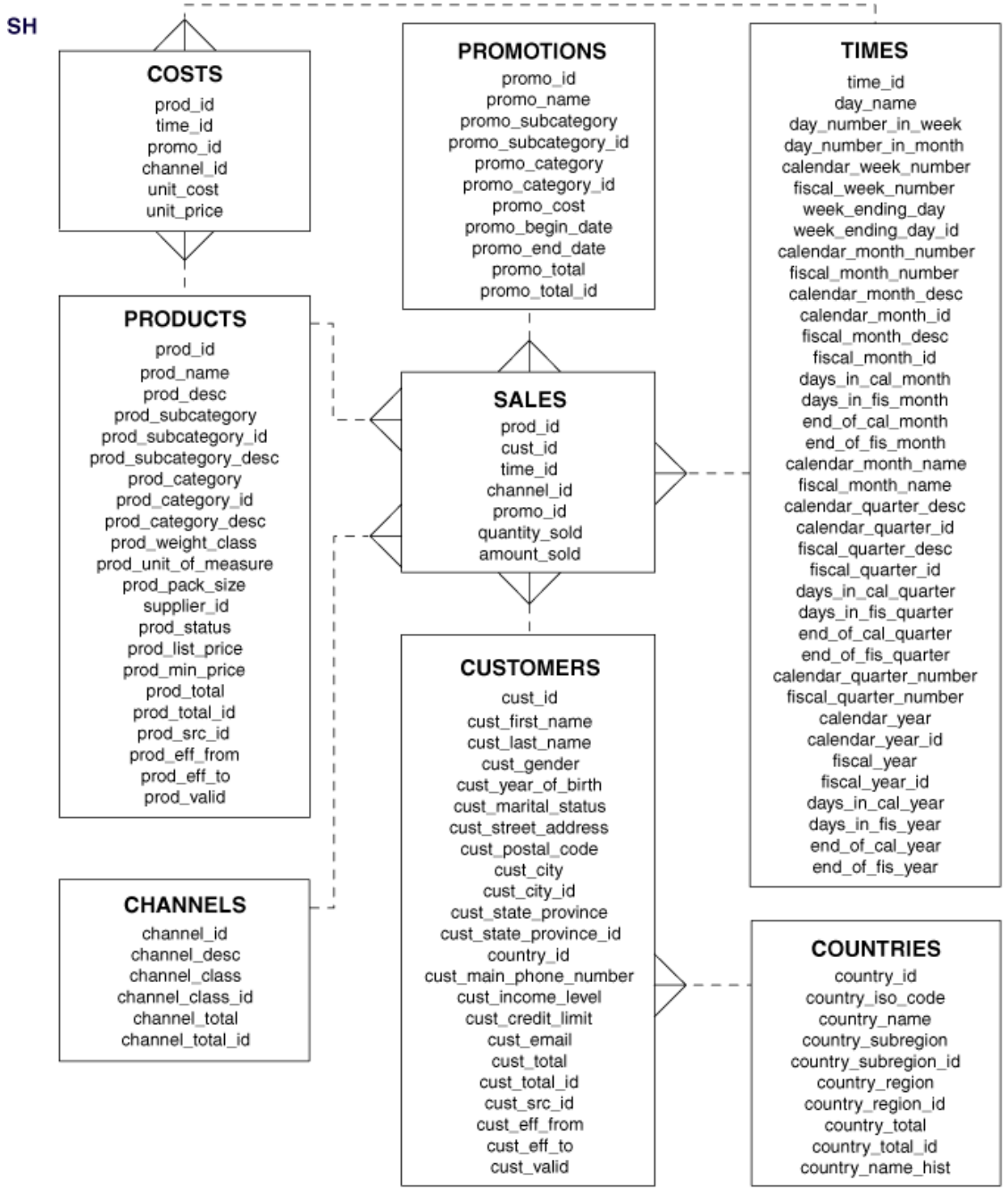
Jupyter is my favourite notebook (for now) but doesn't have a kernel for PGX (maybe will come in the future as Oracle is supposed to support both notebooks in the cloud).

Python will be used to execute PGX commands.

Build a graph from (and in the) database

Sales History (SH)

One of the Oracle Database sample schema



Build a graph from (and in the) database

How to model “this thing” in a graph database ?

Products, Customers, Countries, Sales

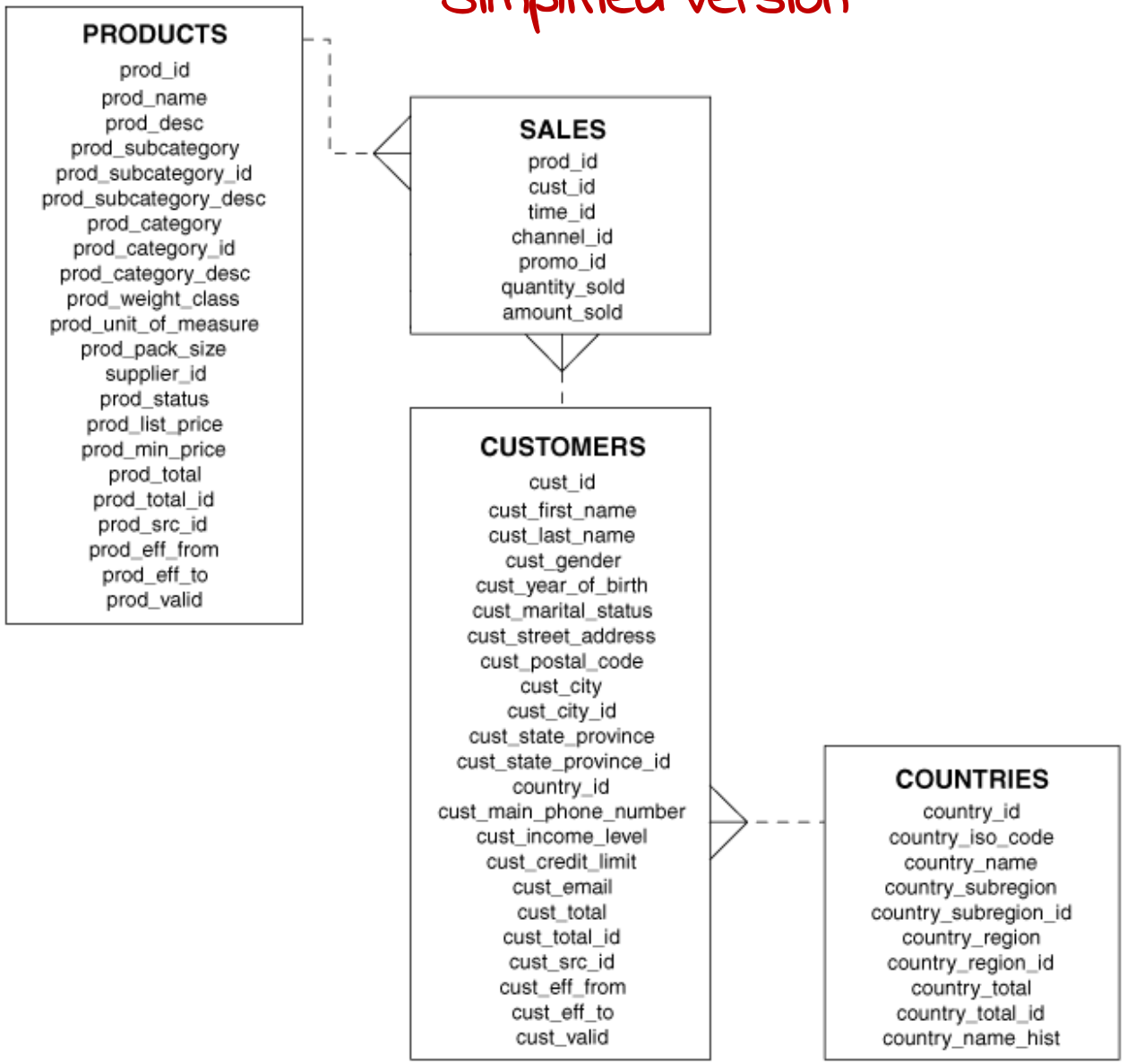
Which nodes to create?

- What is the node ID?
- What is the node label? (if there is one)
- What are the node properties?

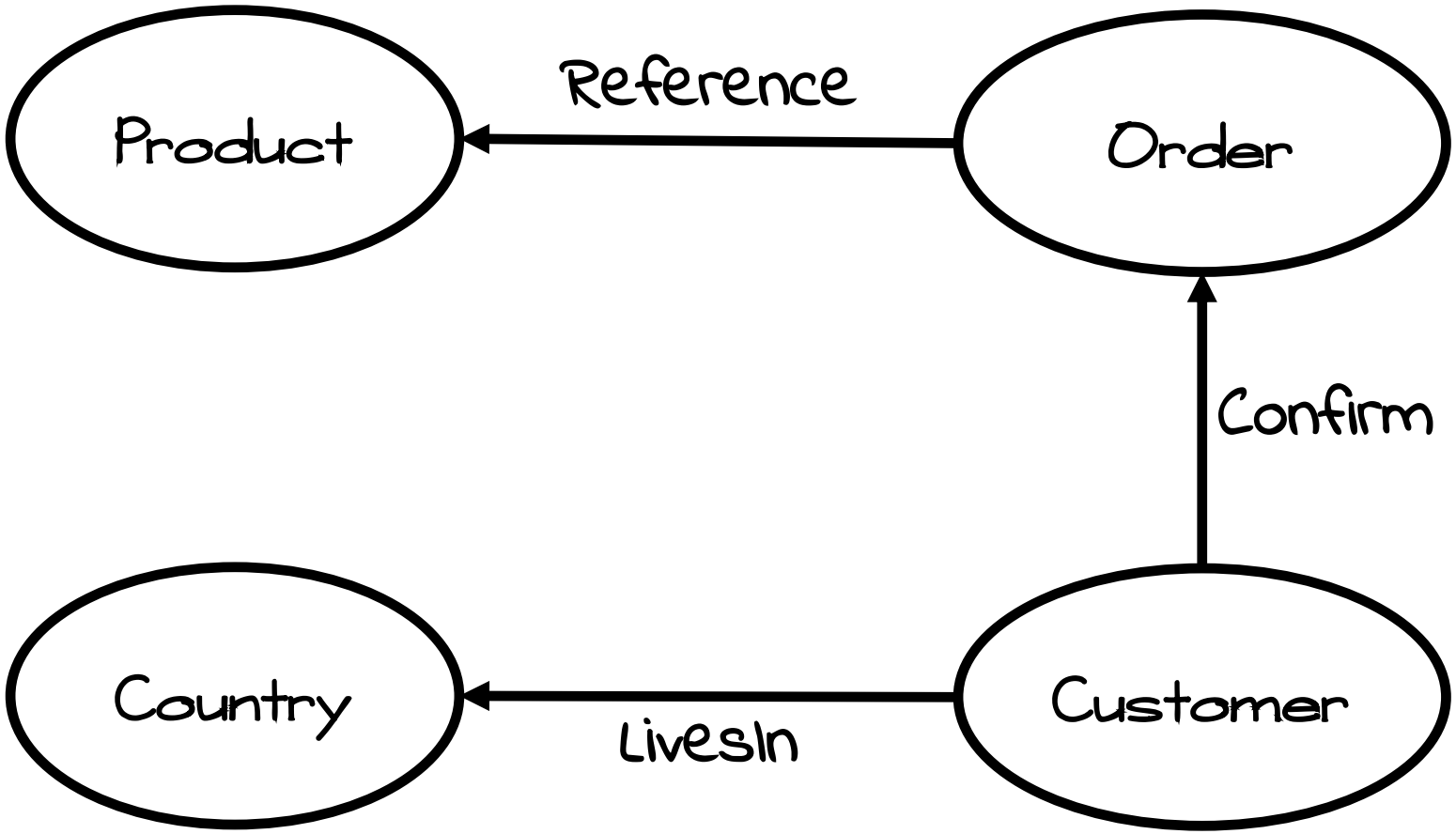
Which edges to create?

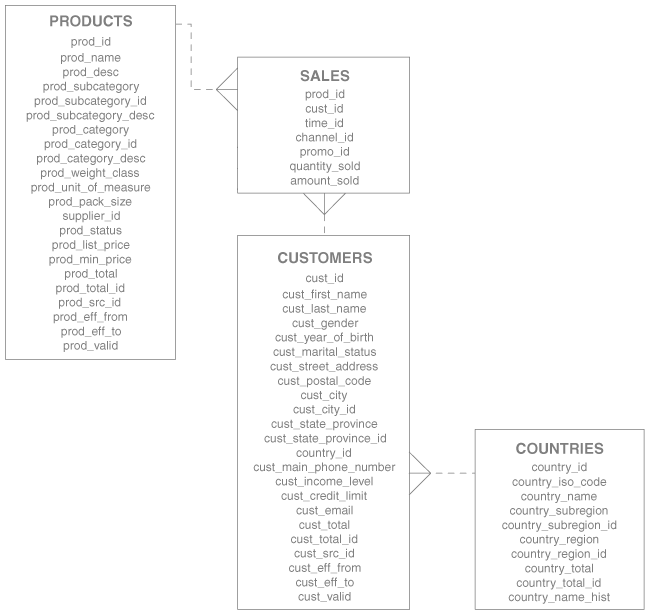
- What is the edge ID?
- What is the edge label? (if there is one)
- What are the edge properties?

Simplified version

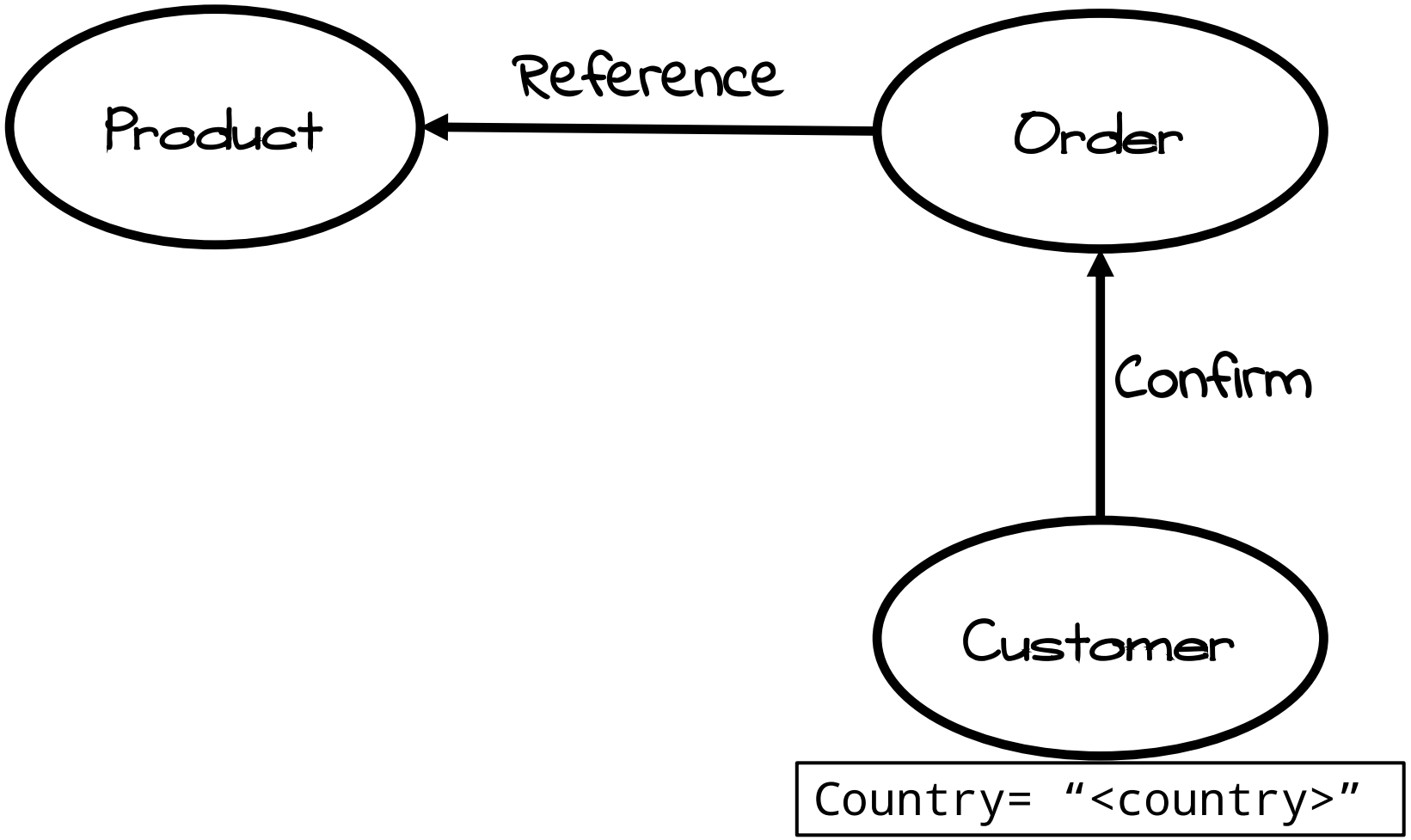


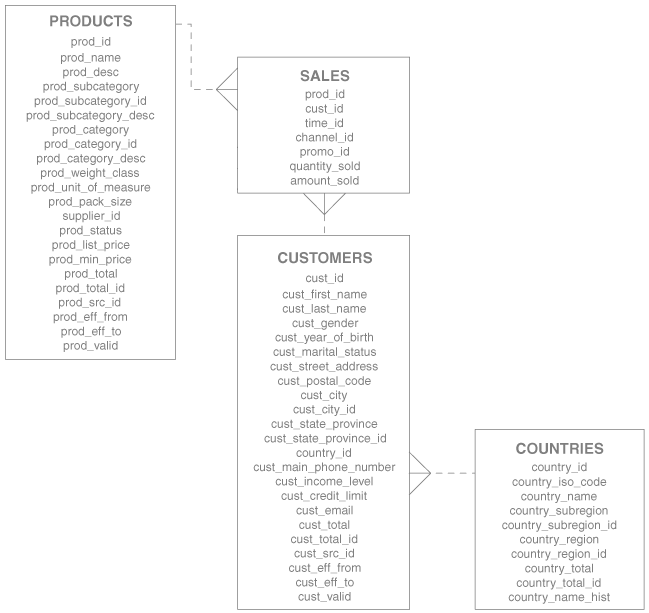
Build a graph from (and in the) database



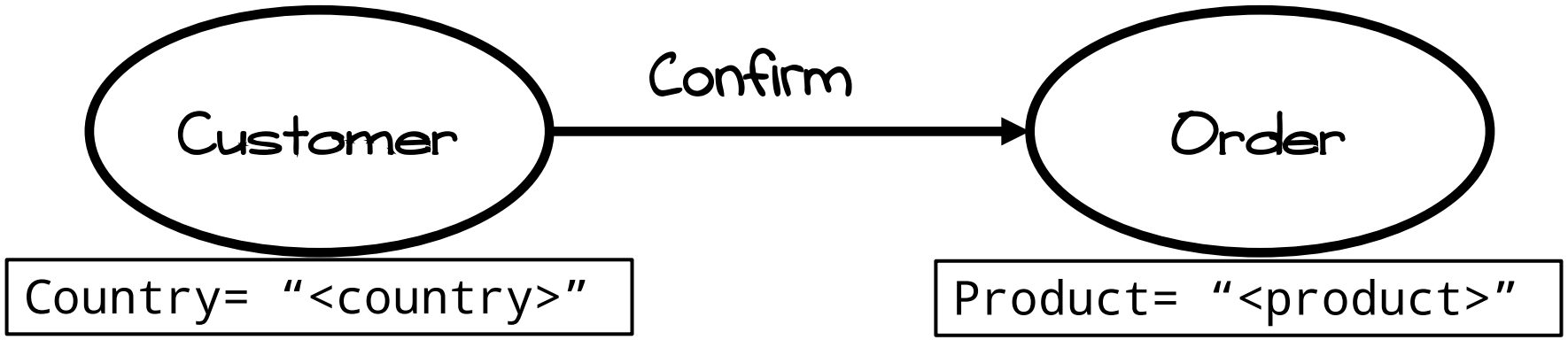


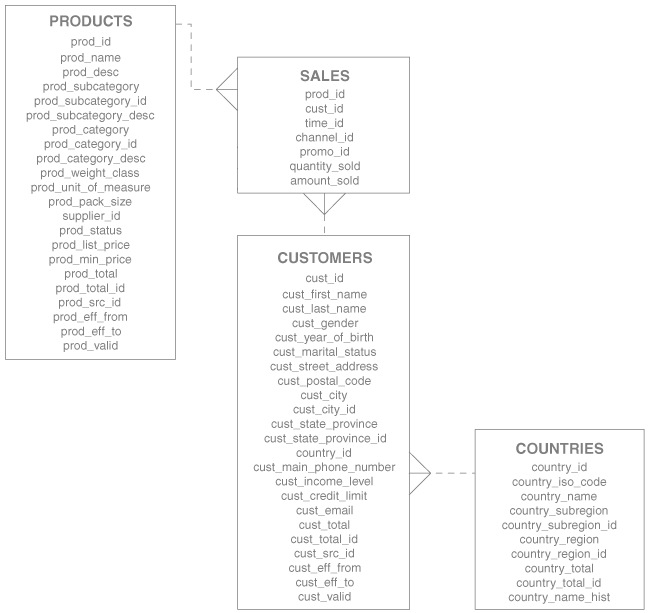
Build a graph from (and in the) database



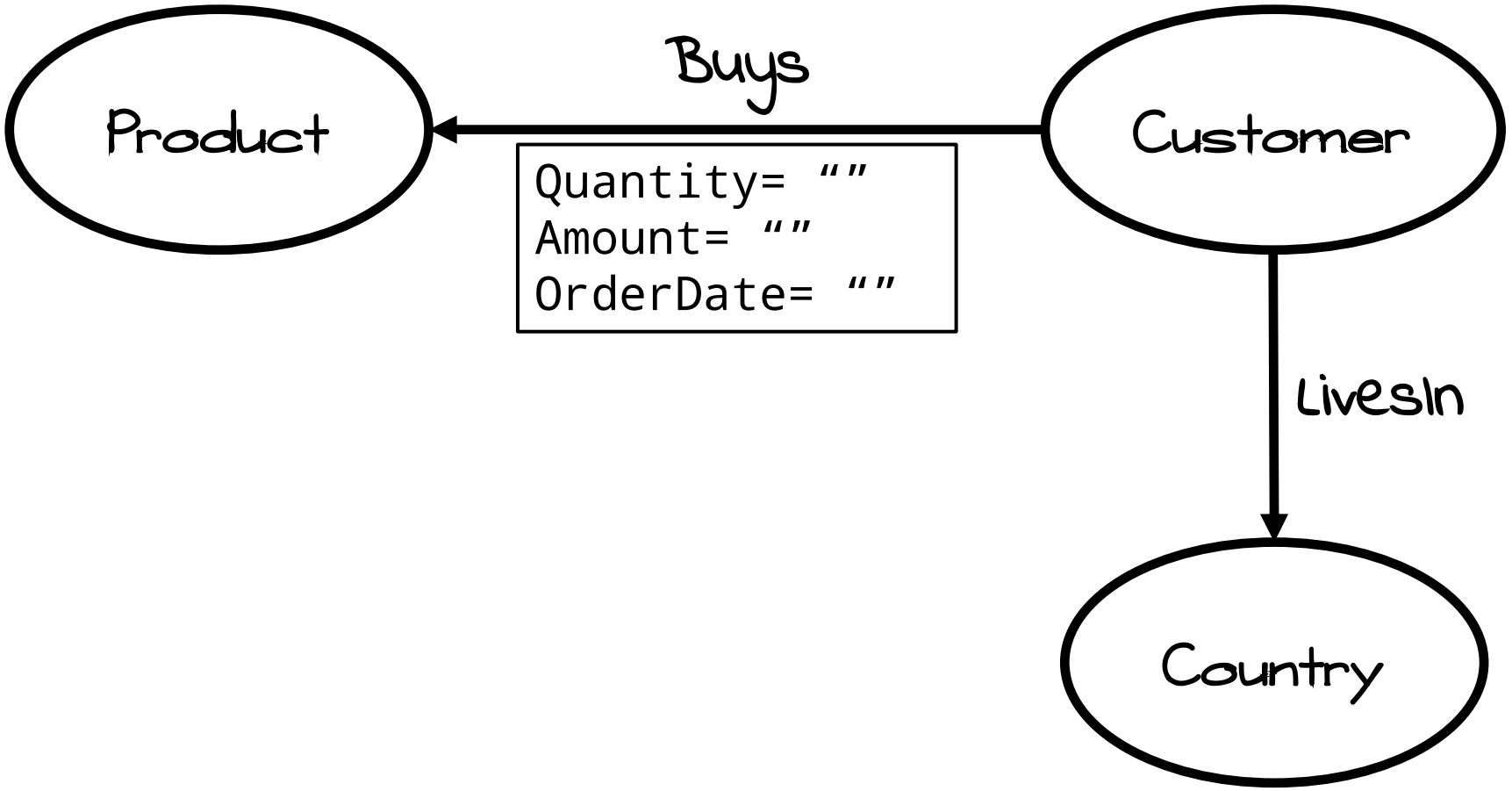


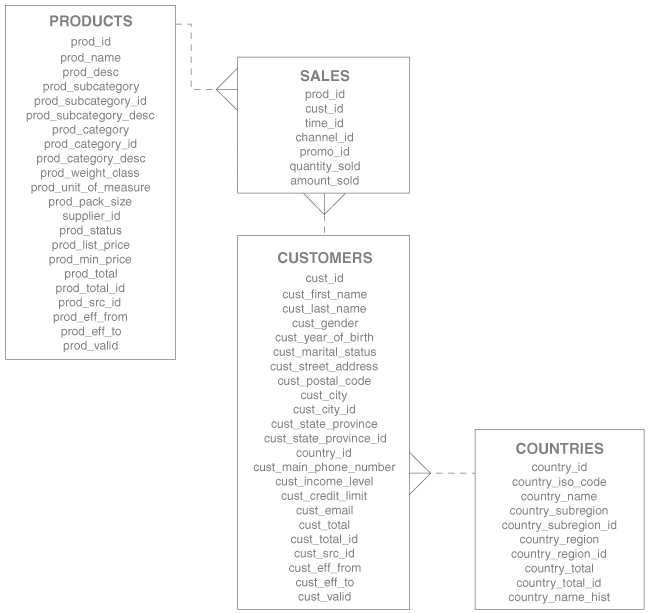
Build a graph from (and in the) database



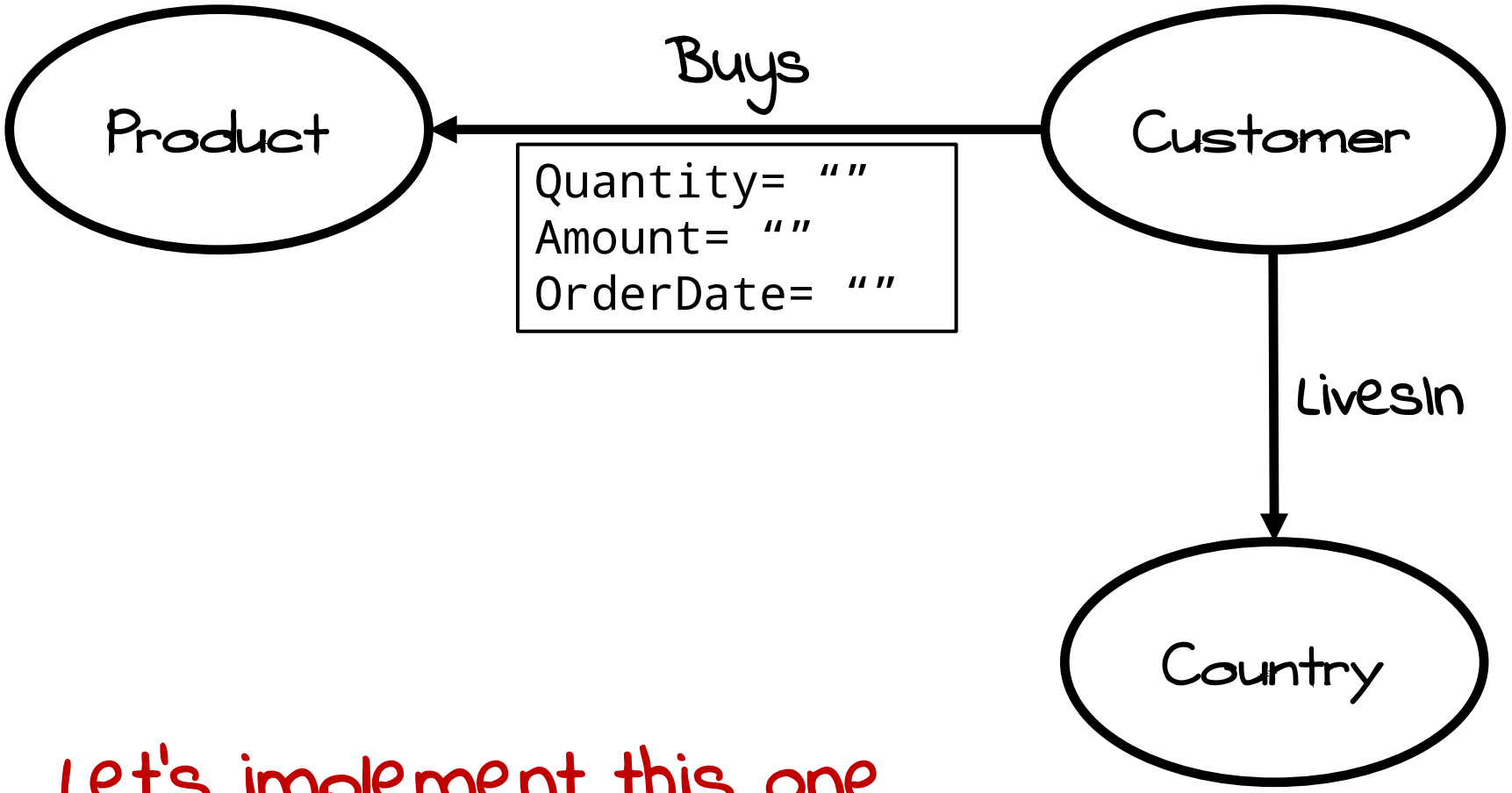


Build a graph from (and in the) database

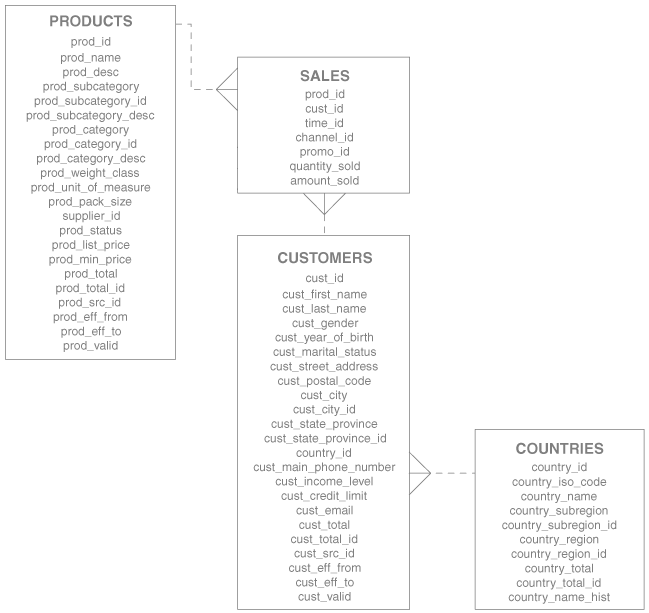




Build a graph from (and in the) database



Let's implement this one



Graph Schema for Oracle Database

Vertex Table: "<graph>VT$"

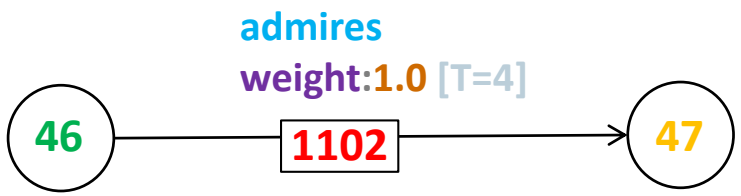
```
Name   Null?    Type
-----  -------- ----------------------
VID    NOT NULL NUMBER
K               NVARCHAR2(3100)
T               INTEGER
V               NVARCHAR2(15000)
VN              NUMBER
VT              TIMESTAMP WITH TIMEZONE
```

47

name: Matthew McConaughey [T=1]
age: 47 [T=2]
birth-date:1969-11-04 12:00:00.0 [T=5]

Edge Table: "<graph>GE$"

```
Name     Null?    Type
-------  -------- ----------------------
EID      NOT NULL NUMBER
SVID     NOT NULL NUMBER
DVID     NOT NULL NUMBER
EL                NVARCHAR2(3100)
K                 NVARCHAR2(3100)
T                 INTEGER
V                 NVARCHAR2(15000)
VN                NUMBER
VT                TIMESTAMP WITH TIME ZONE
```



Graph Schema for Oracle Database

Data Types:

- All numeric properties go in VN
- Date/time properties go in VT
- All others go in V
- Booleans are encoded as "Y" / "N"

NOTE: All numeric and date properties are also stored in V in printable format (to enable text indexing)

ID	Data type	Column
1	String	V
2	Integer	VN
3	Float	VN
4	Double	VN
5	Date	VT
6	Boolean	V
7	Long	VN
8	Short	VN
9	Byte	VN
10	Char	V
101	Serializable	V

Build a graph from (and in the) database

DEMO

Conclusion

Hopefully you see this slide after all the live examples and everything worked as expected.

If not …
Trust me: it really works like when you don't have the "demo effect".

Slides and a PDF version of the notebooks will be delivered to ITOUG, if you want a copy of the notebooks (.ipynb) get in touch and I will send them.

If you feel like having more questions than before this presentation,
it's a good thing!
It means you have to try graphs for real...

In the following pages you find 5 Jupyter notebooks:

- 1 Graph from scratch
- 2 Create graph in database
- 3 Load graph from database
- 4 Save and load graph as file
- 5 Convert PGQL to SQL

Create a file by hand

A graph can be built by hand simply defining nodes and edges one by one with all their properties and labels. This notebook connect to a PGX server instance, but the same works on the standalone PGX shell (./bin/pgx) (not the visualization, just the build and test queries part).

Requirements & Environment

This notebook make use of the *PGX 2.7.0 Server* release available at https://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytix/downloads/index.html (https://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytix/downloads/index.html).
The PGX server is configured to listen on *port 7008* without SSL and authentication is disabled. The ZIP file has been extracted into `/opt` and created the folder `/opt/pgx-2.7.0` . To connect to this PGX server python will use the *PGX 2.7.0 Java client* release available at the above url and extracted into `/opt/pgx-2.7.0-java-client` .
(Beware if extracted in the same location as the *Server* archive, both will try to create the *pgx-2.7.0* folder resulting in one overriding the other. Extract the *Java client* archive first, rename the extracted folder and after you can safely extract the *Server* archive.)
You can extract both these archives in the location of your choice and simply need to adapt the below paths (the one used in the `startJVM` call and the one to start the PGX server.

This Notebook is using python 3.6, but the graph code works the same on python 2.7. *JPype1* is required and can be installed using `pip` (this package is the one connecting python to the JVM where the PGX commands will be executed). graphviz is used to display the graph and can be installed using `pip` , this method will not work with huge graphs because the resulting image will be too big.

Import required packages

In [1]:

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

Setup JVM

Start JVM passing the PGX 2.7.0 classpath

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]:

```
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path('/opt/pgx
-2.7.0-java-client/lib'))
```

Create a session on the PGX 2.7.0 server

Need to start the PGX 2.7.0 server before to continue:

```
/opt/pgx-2.7.0/bin/start-server
```

In [4]:

```
session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7008/", "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 0x7fd8ec092d
a0>
```

Build new graph

In [6]:

```
graph = builder.build()

print(graph)
```

```
PgxGraph[name=anonymous_graph_1,N=7,E=9,created=1549205618869]
```

Visualize the graph

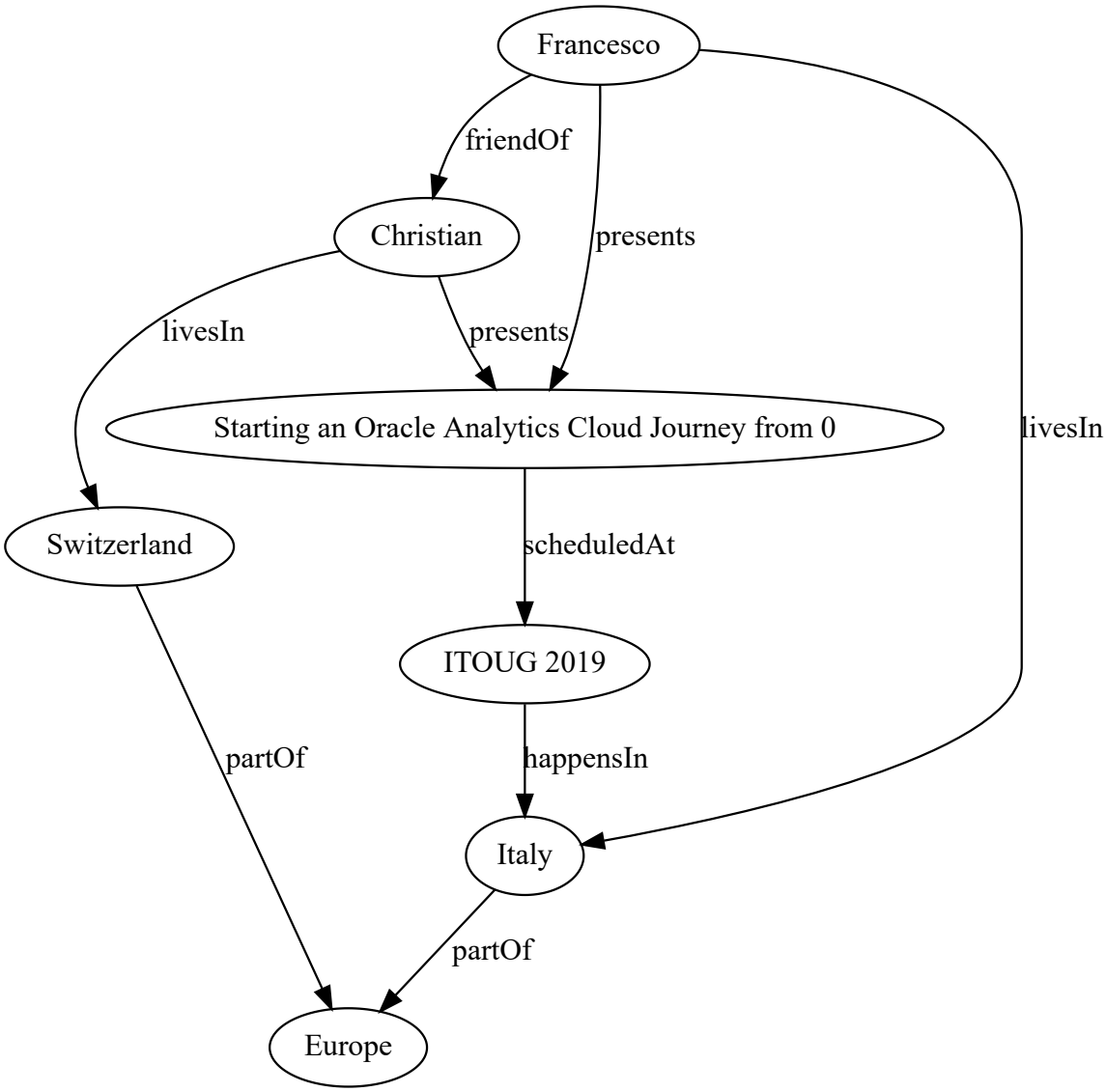
In [7]:

```
# %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]:

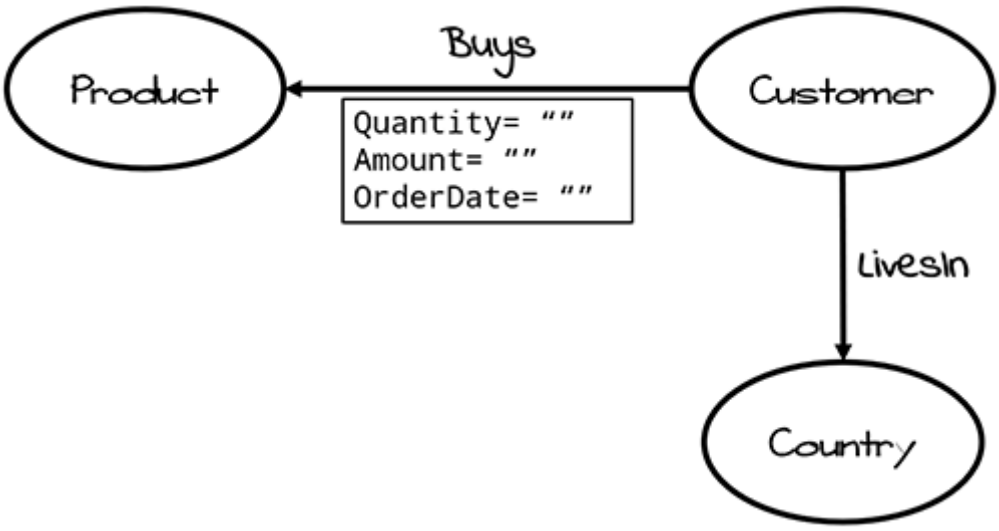


In [ ]:

```
```

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.



Requirements & Environment

This notebook uses the Jupyter magical keyword **%sql** (single line SQL) and **%%sql** (multiple lines SQL) which you can find on https://github.com/catherinedevlin/ipython-sql (https://github.com/catherinedevlin/ipython-sql).
The database is installed on the same host than Jupyter therefore uses `localhost` as hostname, port is the default `1521` and the PDB name is `ORCLPDB1`. The connection is done using `scott` as username and `Admin123` as password. Adapt these settings in all the commands when required to match your environment.

Pandas is used to visualize, analyze and plot the result of some queries, it can be installed using `pip`.

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('itoug', 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', ['itoug', 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 'ITOUG%'
ORDER BY table_name
```

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

Out[3]:

owner	table_name
SCOTT	ITOUGGE$
SCOTT	ITOUGGT$
SCOTT	ITOUGIT$
SCOTT	ITOUGSS$
SCOTT	ITOUGVT$

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" will be used to make sure there is no overlap, simply by addint 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 ITOUGVT$

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

In [8]:

```
%%sql
INSERT INTO ITOUGVT$ (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 ITOUGVT$

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

In [11]:

```
%%sql
INSERT INTO ITOUGVT$ (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 ITOUGVT$

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

In [14]:

```
%%sql
INSERT INTO ITOUGVT$ (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 itougvt$
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 itougvt$
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, therefore there isn't anything on which to build EID (edge ID)

In [17]:

```
%%sql
CREATE SEQUENCE itoug_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 ITOUGGE$

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

In [20]:

```
%%sql
INSERT INTO ITOUGGE$ (eid, svid, dvid, el, k, t, v)
SELECT itoug_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 itoug_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 ITOUGGE$

In [24]:

```
%%sql
INSERT INTO itougge$ (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

In [25]:

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

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

Out[25]:

```
[]
```

In [26]:

```
%%sql
DROP SEQUENCE itoug_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 itougge$
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 itougge$
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.

Requirements & Environment

This notebook make use of PGX embedded into the Oracle Database installation. The database is installed with `ORACLE_HOME = /opt/oracle/product/18c/dbhome_1` , PGX can be found in `$ORACLE_HOME/md/property_graph/pgx/` . The PGX server is configured to listen on *port 7007* without SSL and authentication is disabled.

Pandas is used to visualize, analyze and plot the result of some queries, it can be installed using `pip` .

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]:

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

Create session

Need to start the PGX server before to continue:

```
/opt/oracle/product/18c/dbhome_1/md/property_graph/pgx/bin/start-server
```

In [5]:

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

```
PgxSession[ID=37e2088c-999c-4de2-b615-e5c220929402,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 itougvt$
    UNION ALL
    SELECT DISTINCT k, t, 'Edge' AS kind
    FROM itougge$
)
,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

This version doesn't support DATE as type in PGQL queries results, therefore it's easier to switch them to string (waiting a fix in a future release).

In [7]:

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

cfg = GraphConfigBuilderClass.forPropertyGraphRdbms()\
    .setUsername("scott")\
    .setPassword("Admin123")\
    .setName("itoug")\
    .setMaxNumConnections(4)\
    .setJdbcUrl("jdbc:oracle:thin:@localhost:1521/ORCLPDB1")\
    .setLoadEdgeLabel(True)\
    .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)
```

```
{"vertex_id_type":"long","format":"pg","attributes":{},"db_engine":"RDBM
S","username":"scott","jdbc_url":"jdbc:oracle:thin:@localhost:1521/ORCLPDB
1","name":"itoug","error_handling":{},"loading":{"load_edge_label":tru
e},"edge_props":[{"name":"amount","type":"float"},{"name":"orderDate","typ
e":"string"},{"name":"quantity","type":"float"},{"name":"stateProvince","t
ype":"string"}],"vertex_props":[{"name":"category","type":"string"},{"nam
e":"gender","type":"string"},{"name":"isoCode","type":"string"},{"name":"l
abel","type":"string"},{"name":"listPrice","type":"float"},{"name":"marita
lStatus","type":"string"},{"name":"name","type":"string"},{"name":"sourceI
d","type":"integer"},{"name":"subcategory","type":"string"},{"name":"yearO
fBirth","type":"integer"}],"max_num_connections":4,"password":"SHA-256: [B
@5b1ebf56"}
```

3) Load graph

In [8]:

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

print(opg)
```

```
oraclepropertygraph with name itoug
```

In [9]:

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

print(pgxGraph)
```

```
PgxGraph[name=itoug,N=55595,E=695133,created=1549206288208]
```

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') LIMI
T 10"
        )
pgxResultSet = pgxGraph.queryPgql(query)

print(pgxResultSet)

pgxResults = pgxResultSet.getResults()
for r in pgxResults.iterator():
    print(r.getString(0), 'bought a quantity of', r.getFloat(4), r.getString(1), 'for a
 price of', r.getFloat(3), 'on', r.getString(2))
```

```
PgqlResultSetImpl[graph=itoug,numResults=10]
Anne Koch bought a quantity of 1.0 5MP Telephoto Digital Camera for a pric
e of 1205.99 on 1998-08-05 00:00:00.0
Anne Koch bought a quantity of 2.0 5MP Telephoto Digital Camera for a pric
e of 2411.98 on 1998-10-05 00:00:00.0
Anne Koch bought a quantity of 1.0 5MP Telephoto Digital Camera for a pric
e of 1232.16 on 1998-01-30 00:00:00.0
Rosamond Krider bought a quantity of 2.0 5MP Telephoto Digital Camera for
a price of 2367.85 on 2001-01-17 00:00:00.0
Rosamond Krider bought a quantity of 2.0 5MP Telephoto Digital Camera for
a price of 2059.71 on 2000-10-17 00:00:00.0
Rosamond Krider bought a quantity of 1.0 5MP Telephoto Digital Camera for
a price of 1029.1 on 2000-11-16 00:00:00.0
Raina Silverberg bought a quantity of 1.0 5MP Telephoto Digital Camera for
a price of 1210.21 on 1999-02-20 00:00:00.0
Bertilde Sexton bought a quantity of 1.0 5MP Telephoto Digital Camera for
a price of 1001.7 on 2001-04-20 00:00:00.0
Erica Vandermark bought a quantity of 1.0 5MP Telephoto Digital Camera for
a price of 1058.14 on 2000-06-25 00:00:00.0
Madallyn Ladd bought a quantity of 2.0 5MP Telephoto Digital Camera for a
price of 2140.02 on 1999-01-20 00:00:00.0
```

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)

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

Out[13]:

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

In [14]:

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

Out[14]:

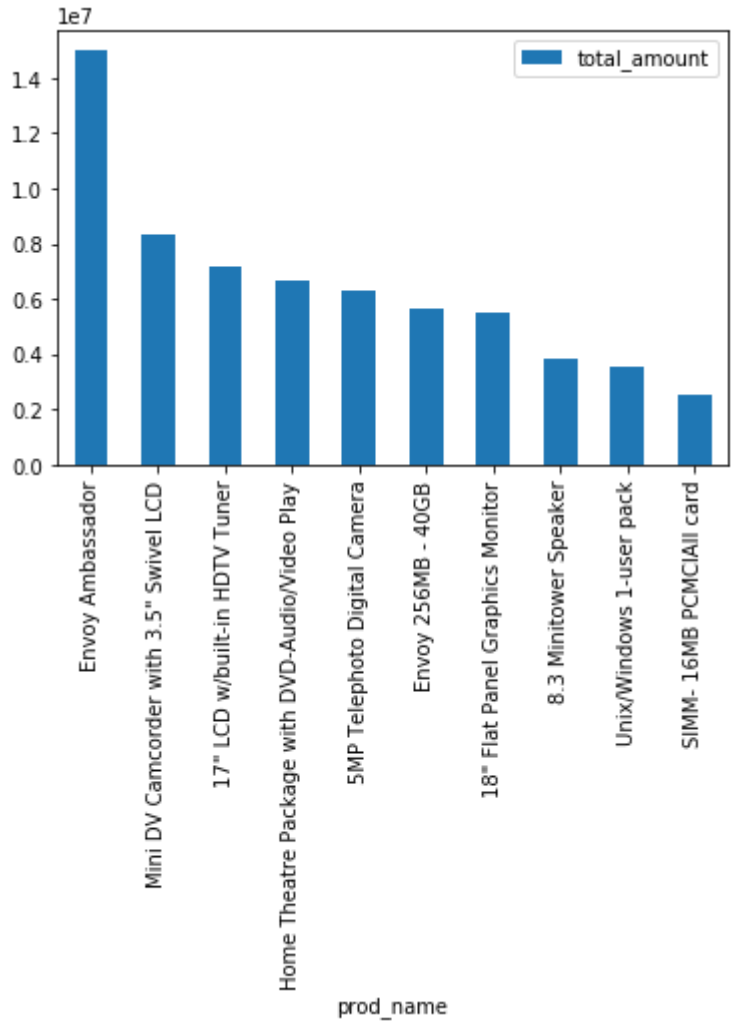
	prod_name	total_quantity	total_amount
count	71	71.000000	7.100000e+01
unique	71	NaN	NaN
top	O/S Documentation Set - English	NaN	NaN
freq	1	NaN	NaN
mean	NaN	12941.450704	1.383181e+06
std	NaN	6024.180200	2.461307e+06
min	NaN	710.000000	2.793333e+04
25%	NaN	8092.000000	2.572828e+05
50%	NaN	12429.000000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

In [16]:

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

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f38cb3b3eb8>
```

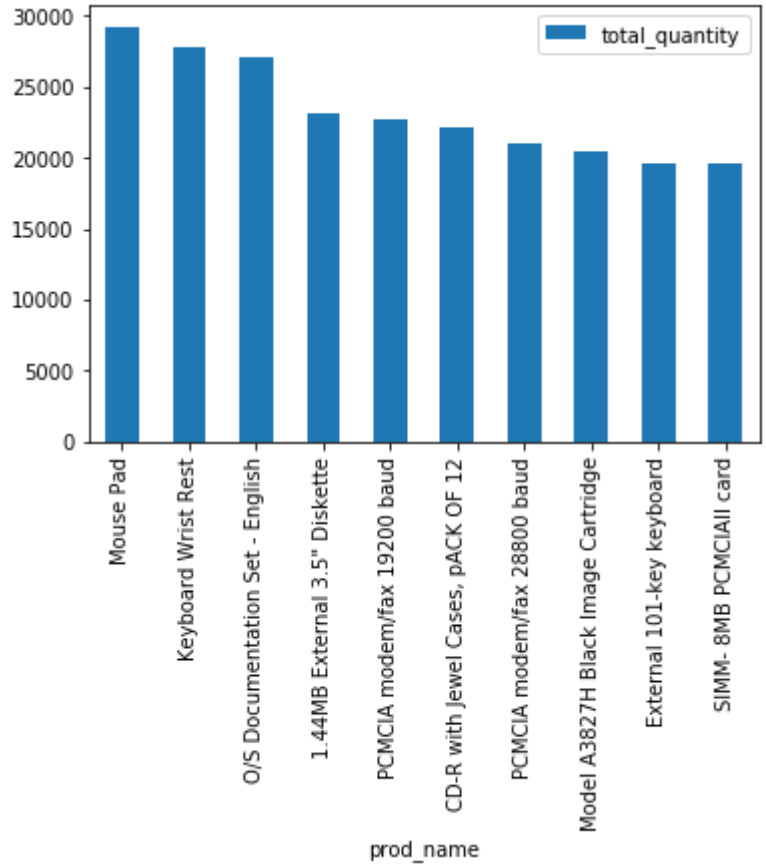


In [17]:

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

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f38cb341630>
```



Analysis with graph algorithms

Run a pagerank algorithm on the graph to find the most important (top pagerank score) nodes

In [18]:

```
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)

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

```
PgqlResultSetImpl[graph=itoug,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 = 5.955638280537223
E-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 = 4.4615106172522
936E-4
product : PCMCIA modem/fax 19200 baud  - page rank = 3.75975313849342E-4
product : 1.44MB External 3.5" Diskette  - page rank = 3.7413260666667993E-
4
product : Standard Mouse  - page rank = 3.7212800945481855E-4
```

Cleanup (free memory now)

In [19]:

```
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('itoug');
END;
```

In [20]:

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

In [ ]:

```
```

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.)
This notebook connect to a PGX server instance, but the same works on the standalone.

Requirements & Environment

This notebook make use of PGX embedded into the Oracle Database installation. The database is installed with `ORACLE_HOME = /opt/oracle/product/18c/dbhome_1` , PGX can be found in `$ORACLE_HOME/md/property_graph/pgx/` . The PGX server is configured to listen on *port 7007* without SSL and authentication is disabled.

This Notebook is using python 3.6, but the graph code works the same on python 2.7. *JPype1* is required and can be installed using `pip` (this package is the one connecting python to the JVM where the PGX commands will be executed). graphviz is used to display the graph and can be installed using `pip` , this method will not work with huge graphs because the resulting image will be too big.

Import required packages

In [1]:

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

Setup JVM

Start JVM passing the PGX 2.7.0 classpath

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]:

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

Create a session

Need to start the PGX server before to continue:

```
/opt/oracle/product/18c/dbhome_1/md/property_graph/pgx/bin/start-server
```

In [4]:

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

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("country", "Italy")
builder.addVertex(2).addLabel("person").setProperty("name", "Christian").setProperty("country", "Switzerland")
builder.addVertex(3).addLabel("session").setProperty("name", "Starting an Oracle Analytics 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 0x7f72ec0a3d
d8>
```

Build new graph

In [6]:

```
graph = builder.build()

print(graph)
```

```
PgxGraph[name=anonymous_graph_4,N=7,E=9,created=1549206372897]
```

Visualize the graph

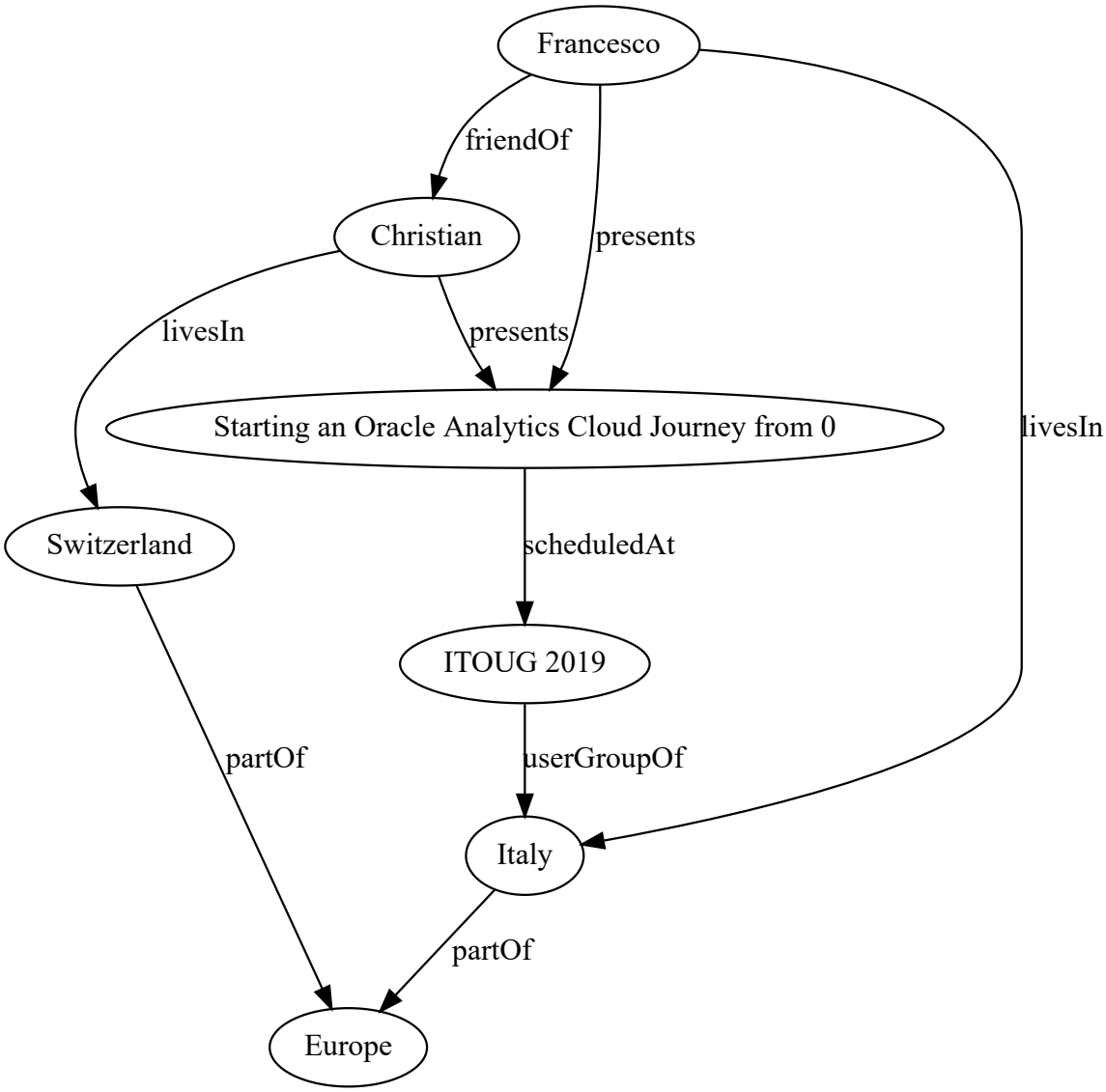
In [7]:

```
# %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]:

```
configFormat = JClass('oracle.pgx.config.Format')

storeIn = '/opt/jupyter/ITOUG_2019/sample_graph.pgb'
storeConfig = graph.store(configFormat.PGB, storeIn, True)
cfgStoreFile = open(storeIn+'.json', 'w')
cfgStoreFile.write(storeConfig.toString())
cfgStoreFile.close()

print(storeConfig)
```

{"format":"pgb","vertex_id_type":"long","attributes":{},"loading":{"load_v
ertex_labels":true,"load_edge_label":true},"error_handling":{},"edge_prop
s":[],"vertex_uris":["/opt/jupyter/ITOUG_2019/sample_graph.pgb"],"vertex_p
rops":[{"name":"name","type":"string"},{"name":"country","type":"strin
g"}],"edge_uris":[]}

Load graph from disk

In [10]:

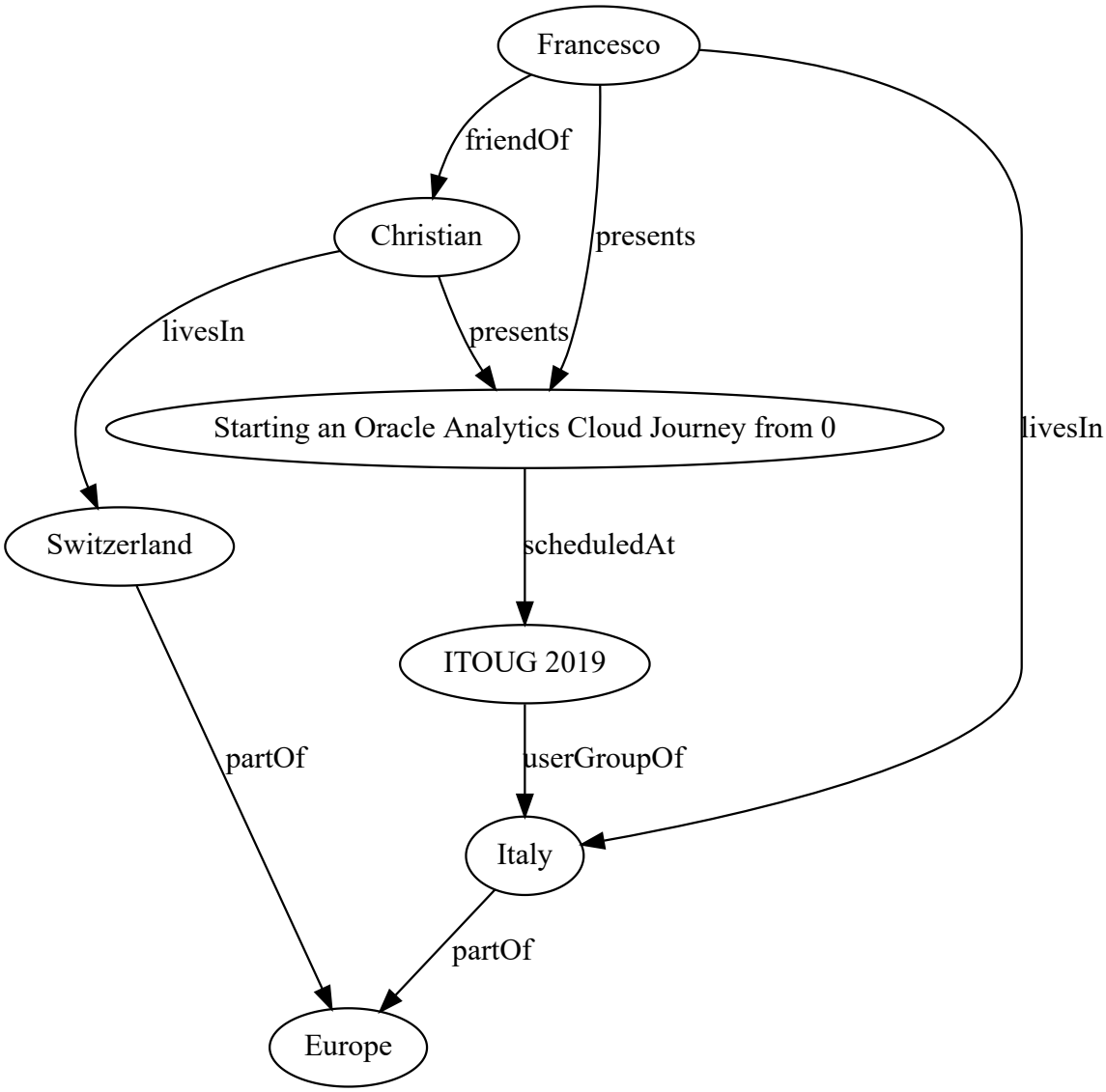
```
myGraph = session.readGraphWithProperties('/opt/jupyter/ITOUG_2019/sample_graph.pgb.jso
n')

print(myGraph)
```

PgxGraph[name=sample_graph,N=7,E=9,created=1549206380395]

In [11]:

```
renderGraph(myGraph)
```

Out[11]:



In [ ]:

Convert PGQL to SQL

There is a method to convert a PGQL to pure SQL which can be execute in the database directly without loading the graph into PGX.

In [1]:

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

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]:

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

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.

In [4]:

```
# 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, 'itoug')

# Execute query to get an OraclePgqlResultSet object
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"
       )

# Create an OraclePgqlStatement
ops = OraclePgqlExecutionFactoryClass.createStatement(opg)
# Get the SQL translation
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 T4.T AS "c.name$T",
T4.V AS "c.name$V",
T4.VN AS "c.name$VN",
T4.VT AS "c.name$VT",
T1.T AS "p.name$T",
T1.V AS "p.name$V",
T1.VN AS "p.name$VN",
T1.VT AS "p.name$VT",
T3.T AS "b.orderDate$T",
T3.V AS "b.orderDate$V",
T3.VN AS "b.orderDate$VN",
T3.VT AS "b.orderDate$VT",
T0.T AS "b.amount$T",
T0.V AS "b.amount$V",
T0.VN AS "b.amount$VN",
T0.VT AS "b.amount$VT",
T2.T AS "b.quantity$T",
T2.V AS "b.quantity$V",
T2.VN AS "b.quantity$VN",
T2.VT AS "b.quantity$VT"
FROM "SCOTT".ITOUGGE$ T0,
"SCOTT".ITOUGVT$ T1,
"SCOTT".ITOUGGE$ T2,
"SCOTT".ITOUGGE$ T3,
"SCOTT".ITOUGVT$ T4
WHERE T0.K=n'amount' AND
T1.K=n'name' AND
T2.K=n'quantity' AND
T3.K=n'orderDate' AND
T4.K=n'name' AND
T0.DVID=T1.VID AND
T0.EID=T2.EID AND
T0.EID=T3.EID AND
T0.SVID=T4.VID AND
(T0.T = 1 AND T0.V = n'product') AND
(T0.T = 1 AND T0.V = n'customer') AND
(T0.EL = n'buys'))
WHERE ROWNUM <= 10
```

Test SQL

In [ ]:

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

In [ ]:

```
%%sql
SELECT * FROM(SELECT T4.T AS "c.name$T",
T4.V AS "c.name$V",
T4.VN AS "c.name$VN",
T4.VT AS "c.name$VT",
T1.T AS "p.name$T",
T1.V AS "p.name$V",
T1.VN AS "p.name$VN",
T1.VT AS "p.name$VT",
T3.T AS "b.orderDate$T",
T3.V AS "b.orderDate$V",
T3.VN AS "b.orderDate$VN",
T3.VT AS "b.orderDate$VT",
T0.T AS "b.amount$T",
T0.V AS "b.amount$V",
T0.VN AS "b.amount$VN",
T0.VT AS "b.amount$VT",
T2.T AS "b.quantity$T",
T2.V AS "b.quantity$V",
T2.VN AS "b.quantity$VN",
T2.VT AS "b.quantity$VT"
FROM "SCOTT".ITOUGGE$ T0,
"SCOTT".ITOUGVT$ T1,
"SCOTT".ITOUGGE$ T2,
"SCOTT".ITOUGGE$ T3,
"SCOTT".ITOUGVT$ T4
WHERE T0.K=n'amount' AND
T1.K=n'name' AND
T2.K=n'quantity' AND
T3.K=n'orderDate' AND
T4.K=n'name' AND
T0.DVID=T1.VID AND
T0.EID=T2.EID AND
T0.EID=T3.EID AND
T0.SVID=T4.VID AND
(T0.T = 1 AND T0.V = n'product') AND
(T0.T = 1 AND T0.V = n'customer') AND
(T0.EL = n'buys'))
WHERE ROWNUM <= 10
```

In [ ]: