From SQL to Graph

It could be possible to create and load the graph fully by SQL as it will be stored in the database itself.
But there are alternatives, likes this bit of Python, which make the work "easier". Mainly the annoying part of generating unique vertices IDs and doing the unpivot to store all the properties as rows (key - values).

Import libraries & connect to the DB

Pandas and **numpy** will be used for data manipulation and transformation. **ipython-sql**, **sqlalchemy** and **Oracle_cx** will be used to connect to the database.

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

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

Out[2]:

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

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('orclsample', 4, 8, '');
END;
```

In [3]:

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

Extract nodes

Query the database to collect all the rows which will represent nodes in the graph with the required properties (as columns).
Some columns will be renamed to have matching columns names with other nodes and some uniformity in nodes structures (mainly database IDs and name). Also add a new column with a hardcoded value representing the `label` of the nodes (which could be a property or the real label of vertices in the graph).

1) Regions

In [4]:

```
%sql result << SELECT region_id, region_name FROM hr.regions

dt = result.DataFrame()
dt.rename(columns={'region_id' : 'id', 'region_name' : 'name'}, inplace=True)
dt['label'] = 'region'

df1 = dt.copy()
dt.head()
```

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

Out[4]:

	id	name	label
0	1	Europe	region
1	2	Americas	region
2	3	Asia	region
3	4	Middle East and Africa	region

2) Countries

In [5]:

```
%sql result << SELECT country_id, country_name FROM hr.countries

dt = result.DataFrame()
dt.rename(columns={'country_id' : 'id', 'country_name' : 'name'}, inplace=True)
dt['label'] = 'country'

df2 = dt.copy()
dt.head()
```

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

Out[5]:

	id	name	label
0	AR	Argentina	country
1	AU	Australia	country
2	BE	Belgium	country
3	BR	Brazil	country
4	CA	Canada	country

3) Locations

In [6]:

```
%sql result << SELECT location_id, city name, city, country_id, state_province, postal_
code, street_address FROM hr.locations

dt = result.DataFrame()
dt.rename(columns={'location_id' : 'id', 'country_id' : 'country', 'street_address' :
'address'}, inplace=True)
dt['label'] = 'location'

df3 = dt.copy()
dt.head()
```

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

Out[6]:

	id	name	city	country	state_province	postal_code	address	label
0	1000	Roma	Roma	IT	None	00989	1297 Via Cola di Rie	location
1	1100	Venice	Venice	IT	None	10934	93091 Calle della Testa	location
2	1200	Tokyo	Tokyo	JP	Tokyo Prefecture	1689	2017 Shinjuku-ku	location
3	1300	Hiroshima	Hiroshima	JP	None	6823	9450 Kamiya-cho	location
4	1400	Southlake	Southlake	US	Texas	26192	2014 Jabberwocky Rd	location

4) Warehouses

In [7]:

```
%sql result << SELECT warehouse_id, warehouse_name FROM oe.warehouses

dt = result.DataFrame()
dt.rename(columns={'warehouse_id' : 'id', 'warehouse_name' : 'name'}, inplace=True)
dt['label'] = 'warehouse'

df4 = dt.copy()
dt.head()
```

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

Out[7]:

	id	name	label
0	1	Southlake, Texas	warehouse
1	2	San Francisco	warehouse
2	3	New Jersey	warehouse
3	4	Seattle, Washington	warehouse
4	5	Toronto	warehouse

5) Departments

In [8]:

```
%sql result << SELECT department_id, department_name FROM hr.departments

dt = result.DataFrame()
dt.rename(columns={'department_id' : 'id', 'department_name' : 'name'}, inplace=True)
dt['label'] = 'department'

df5 = dt.copy()
dt.head()
```

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

Out[8]:

	id	name	label
0	10	Administration	department
1	20	Marketing	department
2	30	Purchasing	department
3	40	Human Resources	department
4	50	Shipping	department

6) Employees

In [9]:

```
%sql result << SELECT employee_id, first_name || ' ' || last_name as name, phone_numbe
r, hire_date, salary FROM hr.employees

dt = result.DataFrame()
dt.rename(columns={'employee_id' : 'id'}, inplace=True)
dt['salary'] = pd.to_numeric(dt['salary'], errors='coerce')
dt['label'] = 'employee'

df6 = dt.copy()
dt.head()
```

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

Out[9]:

	id	name	phone_number	hire_date	salary	label
0	100	Steven King	515.123.4567	2003-06-17	24000.0	employee
1	101	Neena Kochhar	515.123.4568	2005-09-21	17000.0	employee
2	102	Lex De Haan	515.123.4569	2001-01-13	17000.0	employee
3	103	Alexander Hunold	590.423.4567	2006-01-03	9000.0	employee
4	104	Bruce Ernst	590.423.4568	2007-05-21	6000.0	employee

7) Jobs

In [10]:

```
%sql result << SELECT job_id, job_title FROM hr.jobs

dt = result.DataFrame()
dt.rename(columns={'job_id' : 'id', 'job_title' : 'name'}, inplace=True)
dt['label'] = 'role'

df7 = dt.copy()
dt.head()
```

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

Out[10]:

	id	name	label
0	AD_PRES	President	role
1	AD_VP	Administration Vice President	role
2	AD_ASST	Administration Assistant	role
3	FI_MGR	Finance Manager	role
4	FI_ACCOUNT	Accountant	role

8) Products

In [11]:

```
%sql result << SELECT product_id, product_name, product_description, list_price, min_pr
ice FROM oe.product_information

dt = result.DataFrame()
dt.rename(columns={'product_id' : 'id', 'product_name' : 'name', 'product_description'
: 'description'}, inplace=True)
dt['list_price'] = pd.to_numeric(dt['list_price'], errors='coerce')
dt['min_price'] = pd.to_numeric(dt['min_price'], errors='coerce')
dt['label'] = 'product'

df8 = dt.copy()
dt.head()
```

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

Out[11]:

	id	name	description	list_price	min_price	label
0	3091	VRAM - 64 MB	Citrus Video RAM memory module - 64 MB capacit...	279.0	243.0	product
1	1787	CPU D300	Dual CPU @ 300Mhz. For light personal processi...	101.0	90.0	product
2	2439	CPU D400	Dual CPU @ 400Mhz. Good price/performance rati...	123.0	105.0	product
3	1788	CPU D600	Dual CPU @ 600Mhz. State of the art, high cloc...	178.0	149.0	product
4	2375	GP 1024x768	Graphics Processor, resolution 1024 X 768 pixe...	78.0	69.0	product

9) Orders

In [12]:

```
%sql result << SELECT order_id, order_date, order_mode, order_total FROM oe.orders

dt = result.DataFrame()
dt.rename(columns={'order_id' : 'id', 'order_date' : 'date', 'order_mode' : 'mode', 'or
der_total' : 'total'}, inplace=True)
dt['total'] = pd.to_numeric(dt['total'], errors='coerce')
dt['label'] = 'order'

df9 = dt.copy()
dt.head()
```

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

Out[12]:

	id	date	mode	total	label
0	2458	2007-08-16 14:34:12.234359	direct	78279.6	order
1	2397	2007-11-19 15:41:54.696211	direct	42283.2	order
2	2454	2007-10-02 16:49:34.678340	direct	6653.4	order
3	2354	2008-07-14 17:18:23.234567	direct	46257.0	order
4	2358	2008-01-08 18:03:12.654278	direct	7826.0	order

10) Customers

In [13]:

```
%sql result << SELECT customer_id, cust_first_name || ' ' || cust_last_name name, c.cus
t_address.street_address address, c.cust_address.postal_code postal_code, c.cust_addres
s.city city, c.cust_address.state_province state_province, c.cust_address.country_id co
untry, cust_email, date_of_birth, marital_status, gender FROM oe.customers c

dt = result.DataFrame()
dt.rename(columns={'customer_id' : 'id', 'cust_email' : 'email'}, inplace=True)
dt['label'] = 'customer'

df10 = dt.copy()
dt.head()
```

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

Out[13]:

	id	name	address	postal_code	city	state_province	country	
0	232	Donald Hunter	5122 Sinclair Ln	21206	Baltimore	MD	US	Donald.Hunter@(
1	233	Graham Spielberg	680 Bel Air Rd	21014	Bel Air	MD	US	Graham.Spielbe
2	234	Dan Roberts	4301 Ashland Ave	21205	Baltimore	MD	US	Dan.Roberts
3	235	Edward Oates	8004 Stansbury Rd	21222	Baltimore	MD	US	Edward.Oates
4	236	Edward Julius	10209 Yearling Dr	20850	Rockville	MD	US	Edward.Ju

Merge all sets of nodes

Generate a unique **VID** for each node and unpivot into the *v, vn, vt* columns adding the relative *t* column for the datatype.
Pandas had methods to make these activities really easy and quick.

1) Merge all the dataframes and generate VID

In [14]:

```
dtnodes = pd.concat([df1, df2, df3, df4, df5, df6, df7, df8, df9, df10], axis=0, ignore
_index=True, sort=False)
dtnodes.rename(columns={'id' : 'src_id'}, inplace=True)
dtnodes['vid'] = np.arange(len(dtnodes))
dtnodes['vid'] = dtnodes['vid']+1  # make it start from 1
dtnodes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 926 entries, 0 to 925
Data columns (total 22 columns):
src_id            926 non-null object
name              821 non-null object
label             926 non-null object
city              342 non-null object
country           342 non-null object
state_province    269 non-null object
postal_code       333 non-null object
address           342 non-null object
phone_number      107 non-null object
hire_date         107 non-null datetime64[ns]
salary            107 non-null float64
description       288 non-null object
list_price        286 non-null float64
min_price         286 non-null float64
date              105 non-null datetime64[ns]
mode              105 non-null object
total             105 non-null float64
email             319 non-null object
date_of_birth     319 non-null datetime64[ns]
marital_status    319 non-null object
gender            319 non-null object
vid               926 non-null int64
dtypes: datetime64[ns](3), float64(4), int64(1), object(14)
memory usage: 159.2+ KB
```

In [15]:

```
dtnodes.head()
```

Out[15]:

	src_id	name	label	city	country	state_province	postal_code	address	phone_nu
0	1	Europe	region	NaN	NaN	NaN	NaN	NaN	
1	2	Americas	region	NaN	NaN	NaN	NaN	NaN	
2	3	Asia	region	NaN	NaN	NaN	NaN	NaN	
3	4	Middle East and Africa	region	NaN	NaN	NaN	NaN	NaN	
4	AR	Argentina	country	NaN	NaN	NaN	NaN	NaN	

5 rows × 22 columns

2) Numeric and Dates values needs to be 'doubled' into VT and VN columns

3 dataframes are generated to represent the `V`, `VN` and `VT` columns, keeping the same VID for a same node across the dataframes. The dataframes are unpivoted to turn columns into rows.

In [16]:

```
dt_v  = dtnodes.copy()
dt_vn = dtnodes[['vid', 'salary', 'list_price', 'min_price', 'total']].copy()
dt_vt = dtnodes[['vid', 'hire_date', 'date', 'date_of_birth']].copy()

dtv  = dt_v.melt(id_vars=['vid'],  var_name='k', value_name='v')
dtvn = dt_vn.melt(id_vars=['vid'], var_name='k', value_name='vn')
dtvt = dt_vt.melt(id_vars=['vid'], var_name='k', value_name='vt')

print("DTV")
print(dtv.info())
print("\nDTVN")
print(dtvn.info())
print("\nDTVT")
print(dtvt.info())
```

```
DTV
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19446 entries, 0 to 19445
Data columns (total 3 columns):
vid    19446 non-null int64
k      19446 non-null object
v      7073 non-null object
dtypes: int64(1), object(2)
memory usage: 455.8+ KB
None

DTVN
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3704 entries, 0 to 3703
Data columns (total 3 columns):
vid    3704 non-null int64
k      3704 non-null object
vn     784 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 86.9+ KB
None

DTVT
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2778 entries, 0 to 2777
Data columns (total 3 columns):
vid    2778 non-null int64
k      2778 non-null object
vt     531 non-null datetime64[ns]
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 65.2+ KB
None
```

3) Build the unified final dataframe

Join the 3 dataframes together based on the VID and K columns, and generated the `T` column.

In [17]:

```
dt = pd.merge(dtv, dtvn, how='left', on=['vid', 'k'])
dt = pd.merge(dt , dtvt, how='left', on=['vid', 'k'])
dt['t'] = 1
dt.loc[dt['vn'].notnull(), 't'] = 3
dt.loc[dt['vt'].notnull(), 't'] = 5
dnodes = dt[['vid', 'k', 't', 'v', 'vn', 'vt']].sort_values(by=['vid', 'k']).dropna(how
='all', subset=['v', 'vn', 'vt'])

dnodes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7073 entries, 1852 to 5555
Data columns (total 6 columns):
vid    7073 non-null int64
k      7073 non-null object
t      7073 non-null int64
v      7073 non-null object
vn     784 non-null float64
vt     531 non-null datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 386.8+ KB
```

In [18]:

```
dnodes.head()
```

Out[18]:

	vid	k	t	v	vn	vt
1852	1	label	1	region	NaN	NaT
926	1	name	1	Europe	NaN	NaT
0	1	src_id	1	1	NaN	NaT
1853	2	label	1	region	NaN	NaT
927	2	name	1	Americas	NaN	NaT

Test how to retrieve VID

As **VID** has been generated, we need to make sure it is possible to retrieve it's value as the edges will need to translate database ids into VID.
Get the VID for *src_id = 110* and *label = customer* :

In [19]:

```
dtnodes[(dtnodes['label'] == 'customer') & (dtnodes['src_id'] == 110)]['vid']
```

Out[19]:

```
696    697
Name: vid, dtype: int64
```

Extract edges

Query the database to collect all the rows which will represent edges in the graph with the required properties (as columns).
Some columns will be renamed to have matching columns names with other edges and some uniformity in edges structures. Also add a new column with a hardcoded value representing the `label` of the edge.

1) Countries - Regions

In [20]:

```
%sql result << SELECT country_id sid, 'country' sl, region_id did, 'region' dl FROM hr.
countries

dt = result.DataFrame()
dt['label'] = 'locatedIn'

df1 = dt.copy()
dt.head()
```

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

Out[20]:

	sid	sl	did	dl	label
0	AR	country	2	region	locatedIn
1	AU	country	3	region	locatedIn
2	BE	country	1	region	locatedIn
3	BR	country	2	region	locatedIn
4	CA	country	2	region	locatedIn

2) Locations - Countries

In [21]:

```
%sql result << SELECT location_id sid, 'location' sl, country_id did, 'country' dl FROM
hr.locations

dt = result.DataFrame()
dt['label'] = 'locatedIn'

df2 = dt.copy()
dt.head()
```

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

Out[21]:

	sid	sl	did	dl	label
0	1000	location	IT	country	locatedIn
1	1100	location	IT	country	locatedIn
2	1200	location	JP	country	locatedIn
3	1300	location	JP	country	locatedIn
4	1400	location	US	country	locatedIn

3) Warehouses - Locations

In [22]:

```
%sql result << SELECT warehouse_id sid, 'warehouse' sl, location_id did, 'location' dl
 FROM oe.warehouses

dt = result.DataFrame()
dt['label'] = 'locatedIn'

df3 = dt.copy()
dt.head()
```

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

Out[22]:

	sid	sl	did	dl	label
0	1	warehouse	1400	location	locatedIn
1	2	warehouse	1500	location	locatedIn
2	3	warehouse	1600	location	locatedIn
3	4	warehouse	1700	location	locatedIn
4	5	warehouse	1800	location	locatedIn

4) Departments - Locations

In [23]:

```
%sql result << SELECT department_id sid, 'department' sl, location_id did, 'location' d
l FROM hr.departments

dt = result.DataFrame()
dt['label'] = 'locatedIn'

df4 = dt.copy()
dt.head()
```

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

Out[23]:

	sid	sl	did	dl	label
0	10	department	1700	location	locatedIn
1	20	department	1800	location	locatedIn
2	30	department	1700	location	locatedIn
3	40	department	2400	location	locatedIn
4	50	department	1500	location	locatedIn

5) Departments - Employees (managers of departments)

In [24]:

```
%sql result << SELECT department_id sid, 'department' sl, manager_id did, 'employee' dl
FROM hr.departments

dt = result.DataFrame()
dt['label'] = 'managedBy'

df5 = dt.copy()
dt.head()
```

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

Out[24]:

	sid	sl	did	dl	label
0	10	department	200.0	employee	managedBy
1	20	department	201.0	employee	managedBy
2	30	department	114.0	employee	managedBy
3	40	department	203.0	employee	managedBy
4	50	department	121.0	employee	managedBy

6) Employees - Employees (direct managers)

In [25]:

```
%sql result << SELECT employee_id sid, 'employee' sl, manager_id did, 'employee' dl FRO
M hr.employees

dt = result.DataFrame()
dt['label'] = 'managedBy'

df6 = dt.copy()
dt.head()
```

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

Out[25]:

	sid	sl	did	dl	label
0	100	employee	NaN	employee	managedBy
1	101	employee	100.0	employee	managedBy
2	102	employee	100.0	employee	managedBy
3	103	employee	102.0	employee	managedBy
4	104	employee	103.0	employee	managedBy

7) Employees - Departments

In [26]:

```
%sql result << SELECT employee_id sid, 'employee' sl, department_id did, 'department' d
l FROM hr.employees

dt = result.DataFrame()
dt['label'] = 'worksIn'

df7 = dt.copy()
dt.head()
```

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

Out[26]:

	sid	sl	did	dl	label
0	100	employee	90.0	department	worksIn
1	101	employee	90.0	department	worksIn
2	102	employee	90.0	department	worksIn
3	103	employee	60.0	department	worksIn
4	104	employee	60.0	department	worksIn

8) Employees - Jobs

In [27]:

```
%sql result << SELECT employee_id sid, 'employee' sl, job_id did, 'role' dl FROM hr.emp
loyees

dt = result.DataFrame()
dt['label'] = 'worksAs'

df8 = dt.copy()
dt.head()
```

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

Out[27]:

	sid	sl	did	dl	label
0	206	employee	AC_ACCOUNT	role	worksAs
1	205	employee	AC_MGR	role	worksAs
2	200	employee	AD_ASST	role	worksAs
3	100	employee	AD_PRES	role	worksAs
4	101	employee	AD_VP	role	worksAs

9) Customers - Countries

In [28]:

```
%sql result << SELECT customer_id sid, 'customer' sl, c.cust_address.country_id did, 'c
ountry' dl FROM oe.customers c

dt = result.DataFrame()
dt['label'] = 'locatedIn'

df9 = dt.copy()
dt.head()
```

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

Out[28]:

	sid	sl	did	dl	label
0	232	customer	US	country	locatedIn
1	233	customer	US	country	locatedIn
2	234	customer	US	country	locatedIn
3	235	customer	US	country	locatedIn
4	236	customer	US	country	locatedIn

10) Products - Orders

In [29]:

```
%sql result << SELECT product_id sid, 'product' sl, order_id did, 'order' dl, unit_pric
e, quantity FROM oe.order_items

dt = result.DataFrame()
dt['unit_price'] = pd.to_numeric(dt['unit_price'], errors='coerce')
dt['quantity'] = pd.to_numeric(dt['quantity'], errors='coerce')
dt['label'] = 'itemOf'

df10 = dt.copy()
dt.head()
```

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

Out[29]:

	sid	sl	did	dl	unit_price	quantity	label
0	2289	product	2355	order	46.0	200	itemOf
1	2264	product	2356	order	199.1	38	itemOf
2	2211	product	2357	order	3.3	140	itemOf
3	1781	product	2358	order	226.6	9	itemOf
4	2337	product	2359	order	270.6	1	itemOf

11) Customers - Orders

In [30]:

```
%sql result << SELECT customer_id sid, 'customer' sl, order_id did, 'order' dl FROM oe.
orders

dt = result.DataFrame()
dt['label'] = 'submit'

df11 = dt.copy()
dt.head()
```

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

Out[30]:

	sid	sl	did	dl	label
0	101	customer	2458	order	submit
1	101	customer	2447	order	submit
2	101	customer	2413	order	submit
3	101	customer	2430	order	submit
4	102	customer	2397	order	submit

12) Employees - Orders

In [31]:

```
%sql result << SELECT sales_rep_id sid, 'employee' sl, order_id did, 'order' dl FROM o
e.orders WHERE sales_rep_id is not null

dt = result.DataFrame()
dt['label'] = 'submit'

df12 = dt.copy()
dt.head()
```

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

Out[31]:

	sid	sl	did	dl	label
0	153	employee	2458	order	submit
1	153	employee	2414	order	submit
2	153	employee	2422	order	submit
3	153	employee	2424	order	submit
4	153	employee	2453	order	submit

13) Products - Warehouses

In [32]:

```
%sql result << SELECT product_id sid, 'product' sl, warehouse_id did, 'warehouse' dl, q
uantity_on_hand FROM oe.inventories

dt = result.DataFrame()
dt.rename(columns={'quantity_on_hand' : 'quantity'}, inplace=True)
dt['quantity'] = pd.to_numeric(dt['quantity'], errors='coerce')
dt['label'] = 'availableIn'

df13 = dt.copy()
dt.head()
```

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

Out[32]:

	sid	sl	did	dl	quantity	label
0	1733	product	1	warehouse	106	availableIn
1	1734	product	1	warehouse	106	availableIn
2	1737	product	1	warehouse	106	availableIn
3	1738	product	1	warehouse	107	availableIn
4	1745	product	1	warehouse	108	availableIn

Merge all sets of edges

Generate a unique **EID** for each edge and unpivot into the *v, vn, vt* columns adding the relative *t* column for the datatype.
Pandas had methods to make these activities really easy and quick.

1) Merge all the dataframes and generate EID

The label column is renamed to match the database storage name `EL` .

In [33]:

```
dtedges = pd.concat([df1, df2, df3, df4, df5, df6, df7, df8, df9, df10, df11, df12, df1
3], axis=0, ignore_index=True, sort=False)
dtedges['eid'] = np.arange(len(dtedges))
dtedges['eid'] = dtedges['eid']+1  # make it start from 1
dtedges['svid'] = np.nan
dtedges['dvid'] = np.nan
dtedges.rename(columns={'label' : 'el'}, inplace=True)
dtedges.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2703 entries, 0 to 2702
Data columns (total 10 columns):
sid           2703 non-null object
sl            2703 non-null object
did           2685 non-null object
dl            2703 non-null object
el            2703 non-null object
unit_price    665 non-null float64
quantity      1777 non-null float64
eid           2703 non-null int64
svid          0 non-null float64
dvid          0 non-null float64
dtypes: float64(4), int64(1), object(5)
memory usage: 211.2+ KB
```

In [34]:

```
dtedges.head()
```

Out[34]:

	sid	sl	did	dl	el	unit_price	quantity	eid	svid	dvid
0	AR	country	2	region	locatedIn	NaN	NaN	1	NaN	NaN
1	AU	country	3	region	locatedIn	NaN	NaN	2	NaN	NaN
2	BE	country	1	region	locatedIn	NaN	NaN	3	NaN	NaN
3	BR	country	2	region	locatedIn	NaN	NaN	4	NaN	NaN
4	CA	country	2	region	locatedIn	NaN	NaN	5	NaN	NaN

2) Translate nodes references into IDs

The references to the source and destination nodes of the edges needs to be translated into the above generated VIDs.

In [35]:

```
for i in range(dtedges.shape[0]) :
    # find SVID
    t = dtnodes[(dtnodes['label'] == dtedges.loc[i, 'sl']) & (dtnodes['src_id'] == dted
ges.loc[i, 'sid'])]['vid']
    if t.shape[0] > 0:
        dtedges.loc[i, 'svid'] = t.iloc[0]
    # find DVID
    t = dtnodes[(dtnodes['label'] == dtedges.loc[i, 'dl']) & (dtnodes['src_id'] == dted
ges.loc[i, 'did'])]['vid']
    if t.shape[0] > 0:
        dtedges.loc[i, 'dvid'] = t.iloc[0]
```

In [36]:

```
dtedges.head()
```

Out[36]:

	sid	sl	did	dl	el	unit_price	quantity	eid	svid	dvid
0	AR	country	2	region	locatedIn	NaN	NaN	1	5.0	2.0
1	AU	country	3	region	locatedIn	NaN	NaN	2	6.0	3.0
2	BE	country	1	region	locatedIn	NaN	NaN	3	7.0	1.0
3	BR	country	2	region	locatedIn	NaN	NaN	4	8.0	2.0
4	CA	country	2	region	locatedIn	NaN	NaN	5	9.0	2.0

3) Drop missing SVID or DVID

An edge can only ecist if it has a source and destination, in case of missing nodes (data quality or explicitely missing connection) drop the edge.
First check the missing nodes (sources first, destinations after). After drop them.

In [37]:

```
dtedges.loc[dtedges.svid.isna()]
```

Out[37]:

	sid	sl	did	dl	el	unit_price	quantity	eid	svid	dvid

In [38]:

```
dtedges.loc[dtedges.dvid.isna()]
```

Out[38]:

	sid	sl	did	dl	el	unit_price	quantity	eid	svid	dvid
95	120	department	NaN	employee	managedBy	NaN	NaN	96	73.0	NaN
96	130	department	NaN	employee	managedBy	NaN	NaN	97	74.0	NaN
97	140	department	NaN	employee	managedBy	NaN	NaN	98	75.0	NaN
98	150	department	NaN	employee	managedBy	NaN	NaN	99	76.0	NaN
99	160	department	NaN	employee	managedBy	NaN	NaN	100	77.0	NaN
100	170	department	NaN	employee	managedBy	NaN	NaN	101	78.0	NaN
101	180	department	NaN	employee	managedBy	NaN	NaN	102	79.0	NaN
102	190	department	NaN	employee	managedBy	NaN	NaN	103	80.0	NaN
103	200	department	NaN	employee	managedBy	NaN	NaN	104	81.0	NaN
104	210	department	NaN	employee	managedBy	NaN	NaN	105	82.0	NaN
105	220	department	NaN	employee	managedBy	NaN	NaN	106	83.0	NaN
106	230	department	NaN	employee	managedBy	NaN	NaN	107	84.0	NaN
107	240	department	NaN	employee	managedBy	NaN	NaN	108	85.0	NaN
108	250	department	NaN	employee	managedBy	NaN	NaN	109	86.0	NaN
109	260	department	NaN	employee	managedBy	NaN	NaN	110	87.0	NaN
110	270	department	NaN	employee	managedBy	NaN	NaN	111	88.0	NaN
111	100	employee	NaN	employee	managedBy	NaN	NaN	112	89.0	NaN
296	178	employee	NaN	department	worksIn	NaN	NaN	297	167.0	NaN
662	782	customer	TH	country	locatedIn	NaN	NaN	663	838.0	NaN
730	378	customer	TH	country	locatedIn	NaN	NaN	731	906.0	NaN

In [39]:

```
dtedges.dropna(how='any', subset=['svid', 'dvid'], inplace=True)
```

4) Drop unused columns

As the translation into `SVID` and `DVID` columns is done, we can drop the temporary columns used to store old references.

In [40]:

```
dtedges.drop(['sid', 'sl', 'did', 'dl'], axis=1, inplace=True)
```

In [41]:

```
dtedges.head()
```

Out[41]:

	el	unit_price	quantity	eid	svid	dvid
0	locatedIn	NaN	NaN	1	5.0	2.0
1	locatedIn	NaN	NaN	2	6.0	3.0
2	locatedIn	NaN	NaN	3	7.0	1.0
3	locatedIn	NaN	NaN	4	8.0	2.0
4	locatedIn	NaN	NaN	5	9.0	2.0

5) Numeric values needs to be 'doubled' into VN column

2 dataframes are generated to represent the `V` and `VN` columns, keeping the same EID for a same edge across the dataframes. The dataframes are unpivoted to turn columns into rows.

In [42]:

```
dt_v  = dtedges.copy()
dt_vn = dtedges[['eid', 'svid', 'dvid', 'el', 'unit_price', 'quantity']].copy()
dt_vn.dropna(how='all', subset=['unit_price', 'quantity'], inplace=True)

dtv  = dt_v.melt(id_vars=['eid', 'svid', 'dvid', 'el'],  var_name='k', value_name='v')
dtvn = dt_vn.melt(id_vars=['eid', 'svid', 'dvid', 'el'], var_name='k', value_name='vn')

print("DTV")
print(dtv.info())
print("\nDTVN")
print(dtvn.info())
```

```
DTV
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5366 entries, 0 to 5365
Data columns (total 6 columns):
eid     5366 non-null int64
svid    5366 non-null float64
dvid    5366 non-null float64
el      5366 non-null object
k       5366 non-null object
v       2442 non-null float64
dtypes: float64(3), int64(1), object(2)
memory usage: 251.6+ KB
None

DTVN
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3554 entries, 0 to 3553
Data columns (total 6 columns):
eid     3554 non-null int64
svid    3554 non-null float64
dvid    3554 non-null float64
el      3554 non-null object
k       3554 non-null object
vn      2442 non-null float64
dtypes: float64(3), int64(1), object(2)
memory usage: 166.7+ KB
None
```

6) Build the unified final dataframe

Join the 2 dataframes together based on the EID, SVID, DVID, EL and K columns, and generated the `T` column.

In [43]:

```
dt = pd.merge(dtv, dtvn, how='left', on=['eid', 'svid', 'dvid', 'el', 'k'])
dt.loc[dt['v'].notnull(), 't'] = 1
dt.loc[dt['vn'].notnull(), 't'] = 3
dt.loc[dt['t'].isna(), 'k'] = np.nan
dt.drop_duplicates(inplace=True)

dedges = dt[['eid', 'svid', 'dvid', 'el', 'k', 't', 'v', 'vn']].sort_values(by=['eid',
'k'])
dedges['v'] = dedges['v'].astype(str)

dedges.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4460 entries, 0 to 2682
Data columns (total 8 columns):
eid     4460 non-null int64
svid    4460 non-null float64
dvid    4460 non-null float64
el      4460 non-null object
k       2442 non-null object
t       2442 non-null float64
v       4460 non-null object
vn      2442 non-null float64
dtypes: float64(4), int64(1), object(3)
memory usage: 313.6+ KB
```

In [44]:

```
dedges.head()
```

Out[44]:

	eid	svid	dvid	el	k	t	v	vn
0	1	5.0	2.0	locatedIn	NaN	NaN	nan	NaN
1	2	6.0	3.0	locatedIn	NaN	NaN	nan	NaN
2	3	7.0	1.0	locatedIn	NaN	NaN	nan	NaN
3	4	8.0	2.0	locatedIn	NaN	NaN	nan	NaN
4	5	9.0	2.0	locatedIn	NaN	NaN	nan	NaN

In [45]:

```
dedges.describe()
```

Out[45]:

	eid	svid	dvid	t	vn
count	4460.000000	4460.000000	4460.000000	2442.0	2442.000000
mean	1515.667489	364.388789	226.723318	3.0	102.016503
std	731.864344	164.963421	234.798412	0.0	185.938139
min	1.000000	5.000000	1.000000	3.0	0.000000
25%	943.750000	261.000000	56.000000	3.0	27.000000
50%	1585.500000	355.000000	60.000000	3.0	61.500000
75%	2146.000000	436.000000	528.000000	3.0	129.000000
max	2703.000000	926.000000	607.000000	3.0	2866.600000

Insert into DB

Store the generated dataframes representing the graph into the database

In [46]:

```
def genSql_insert_from_df(tblName, dtframe):
    '''Generate the SQL statement to insert a dataframe in a table

    Args:
        tblName (string): Name of the table
        dtframe (dataFrame): DataFrame defining the structure

    Returns:
        string: SQL insert statement
    '''
    qry = "INSERT INTO {} (".format(tblName.upper())
    qry = qry + ', '.join(dtframe.columns.to_list()).upper()
    qry = qry + ") VALUES ("
    qry = qry + ', '.join([":{}".format(i+1) for i in range(dtframe.shape[1])])
    qry = qry + ')'
    return qry

def insert_df_in_db(con, tblName, dtframe) :
    '''Insert a dataframe in the database.

    Args:
        con (connection): Database connection
        tblName (string): Name of the table
        dtframe (dataFrame): Dataframe to be inserted

    Returns:
        None
    '''
    # loop variable and batch size
    i = 0
    batch_size = 10000
    # cursor
    cur = con.cursor()

    qry = genSql_insert_from_df(tblName, dtframe)

    # process
    while ((i * batch_size) < len(dtframe)):
        rows = []
        min = i*batch_size
        max = ((i+1)*batch_size)-1
        for x in dtframe.iloc[min:max,:].values:
            rows.append([None if pd.isnull(y) else y for y in x])
        for r in rows:
            cur.execute(qry, r)
#         try:
#             cur.executemany(qry, rows)
#         except cx_Oracle.DatabaseError as e:
#             errorObj, = e.args
#             print("Row", cursor.rowcount, "has error", errorObj.message)

        #cur.executemany(qry, rows)
        i = i + 1

    # commit
    con.commit()

    # close cursor
    cur.close()
```

In [47]:

```
import cx_Oracle
import os
os.environ['NLS_LANG'] = '.AL32UTF8'
```

In [48]:

```
con = cx_Oracle.connect('scott/Admin123@localhost:1521/ORCLPDB1', encoding = "UTF-8", n
encoding = "UTF-8")
cur = con.cursor()
```

In [49]:

```
insert_df_in_db(con, 'orclsamplevt$', dnodes)
insert_df_in_db(con, 'orclsamplege$', dedges)
```

In [50]:

```
cur.close()
con.close()
```

The graph is now stored into the database.

Graph Analysis and translation of PGQL to SQL

The graph has been generated in the database. It can be loaded into PGX for analysis. In this notebook the graph will be loaded from a file for practical reasons, the details on how to load it from the database are available at page 26 of https://gianniceresa.com/wp-content/uploads/2019/05/Graph-Database-from-scratch-in-45-minutes-by-Gianni-Ceresa-Notebooks.pdf (https://gianniceresa.com/wp-content/uploads/2019/05/Graph-Database-from-scratch-in-45-minutes-by-Gianni-Ceresa-Notebooks.pdf) .

In [1]:

```
from jpype import *
import os

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)

#pgxLibsPath = '/opt/pgx-19.1.0-java-client/lib'
pgxLibsPath = '/opt/oracle/product/18c/dbhome_1/md/property_graph/lib/'
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path(pgxLibsPath))
```

In [2]:

```
#session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7008/", "my_session1")
session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7007/", "my_session1")
print(session)
```

```
PgxSession[ID=dc481e79-2c69-4a76-b739-25f81b3d9869,source=my_session1]
```

Load the graph into PGX and publish it

The publish step is only required if another session (or tool) needs to be able to access the graph without going through the loading again. It's a way to share the graph in PGX.

In [3]:

```
graph = session.readGraphWithProperties('/home/oracle/orclsampledb.pgb.json')
graph.publish(graph.getVertexProperties(), graph.getEdgeProperties())
print(graph)
```

```
PgxGraph[name=orclsampledb,N=926,E=2683,created=1558086477861]
```

In [4]:

```
print(session.getGraphs())
```

```
{orclsampledb=PgxGraph[name=orclsampledb,N=926,E=2683,created=155808647786
1]}
```

In [5]:

```
g = session.getGraph('orclsampledb')
print(g)
```

```
PgxGraph[name=orclsampledb,N=926,E=2683,created=1558086477861]
```

Do some analysis by running a PGQL query

Find useful insights by using PGQL queries. This example will sum and count all the orders passed by employees at their managers level. For any level of management (middle managers and up in the hierarchy till the CEO).

In [6]:

```
query = ("SELECT b.name, SUM(o.total), COUNT(DISTINCT id(e)) "
         "MATCH (e) -/:managedBy+/-> (b) "
         ", (e)-[s:submit]->(o) "
         "WHERE e.label = 'employee' AND b.label = 'employee' AND o.label = 'order' "
         "GROUP BY b.name "
         "ORDER BY b.name")
```

In [7]:

```
pgxResultSet = g.queryPgql(query)
print(pgxResultSet)
```

```
PgqlResultSetImpl[graph=orclsampledb,numResults=4]
```

In [8]:

```
pgxResults = pgxResultSet.getResults()
for r in pgxResults.iterator():
    print("{} manages {} employees who submitted orders for a total of {:,}$"
          .format(r.getString(0), r.getLong(2), round(r.getDouble(1)))
         )
```

```
Alberto Errazuriz manages 1 employees who submitted orders for a total of
128,249$
John Russell manages 3 employees who submitted orders for a total of 420,6
04$
Karen Partners manages 5 employees who submitted orders for a total of 1,2
60,054$
Steven King manages 9 employees who submitted orders for a total of 1,808,
907$
```

The 4 rows of results shows 3 mid-managers in the first 3 rows (Alberto, John and Karen) and the CEO (Steven). The numbers of orders and amounts for Steven are the total of the previous 3 rows.
The PGQL query had no idea of how many levels of hierarchies the graph had, it simply got all of them by using the pattern `-/:managedBy+/->` : retrieve connection with 1 or more edges with label *managedBy* connecting nodes.

Translate PGQL into SQL

As found in the documentation there are Java classes to translate a PGQL query into a SQL query which could be executed in the database directly without needing PGX.

In [9]:

```
# 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 (in the DB)
opg = OraclePropertyGraphClass.getInstance(oracle, 'orclsample')
# PGQL query to translate
pgql  = ("SELECT b.name, SUM(o.total), COUNT(DISTINCT id(e)) "
         "MATCH (e) -/:managedBy+/-> (b) "
         ", (e)-[s:submit]->(o) "
         "WHERE e.label = 'employee' AND b.label = 'employee' AND o.label = 'order' "
         "GROUP BY b.name "
         "ORDER BY b.name")
# Create an OraclePgqlStatement
ops = OraclePgqlExecutionFactoryClass.createStatement(opg)
# Get the SQL translation
sqlTrans = ops.translateQuery(pgql, "USE_GT_TAB=T")
print(pgql)
print('----------------------------------------')
print(sqlTrans.getSqlTranslation())
```

```
SELECT b.name, SUM(o.total), COUNT(DISTINCT id(e)) MATCH (e) -/:managedBy
+/-> (b) , (e)-[s:submit]->(o) WHERE e.label = 'employee' AND b.label = 'e
mployee' AND o.label = 'order' GROUP BY b.name ORDER BY b.name
----------------------------------------
SELECT T0$2.T AS "b.name$T",
T0$2.V AS "b.name$V",
T0$2.VN AS "b.name$VN",
T0$2.VT AS "b.name$VT",
4 AS "SUM(o.total)$T",
to_nchar(SUM(T0$4.VN),'TM9','NLS_Numeric_Characters=''.,''') AS "SUM(o.tot
al)$V",
SUM(T0$4.VN) AS "SUM(o.total)$VN",
to_timestamp_tz(null) AS "SUM(o.total)$VT",
7 AS "COUNT(DISTINCT id(e))$T",
to_nchar(COUNT(DISTINCT T0$0.SVID),'TM9','NLS_Numeric_Characters=''.,''')
AS "COUNT(DISTINCT id(e))$V",
COUNT(DISTINCT T0$0.SVID) AS "COUNT(DISTINCT id(e))$VN",
to_timestamp_tz(null) AS "COUNT(DISTINCT id(e))$VT"
FROM (/*Path[*/SELECT DISTINCT SVID, DVID
FROM (
SELECT SVID,DVID FROM
(WITH RW (ROOT, SVID, DVID) AS
( SELECT ROOT, SVID, DVID FROM
(SELECT SVID ROOT, SVID, DVID
FROM (SELECT T0$0.SVID AS SVID,
T0$0.DVID AS DVID
FROM "SCOTT".ORCLSAMPLEGT$ T0$0,
(SELECT DISTINCT VID, VL FROM "SCOTT".ORCLSAMPLEVT$) T0$1
WHERE T0$0.SVID=T0$1.VID AND
(T0$0.EL = n'managedBy' AND T0$0.EL IS NOT NULL) AND
(T0$1.T = 1 AND T0$1.V = n'employee'))
) UNION ALL
SELECT RW.ROOT, R.SVID, R.DVID
FROM (SELECT T0$0.SVID AS SVID,
T0$0.DVID AS DVID
FROM "SCOTT".ORCLSAMPLEGT$ T0$0
WHERE (T0$0.EL = n'managedBy' AND T0$0.EL IS NOT NULL)) R, RW
WHERE RW.DVID = R.SVID )
CYCLE DVID SET cycle_col TO 1 DEFAULT 0
SELECT ROOT SVID, DVID FROM RW))/*]Path*/) T0$0,
"SCOTT".ORCLSAMPLEGT$ T0$1,
"SCOTT".ORCLSAMPLEVT$ T0$2,
(SELECT DISTINCT VID, VL FROM "SCOTT".ORCLSAMPLEVT$) T0$3,
"SCOTT".ORCLSAMPLEVT$ T0$4
WHERE T0$2.K=n'name' AND
T0$4.K=n'total' AND
T0$0.DVID=T0$2.VID AND
T0$0.SVID=T0$1.SVID AND
T0$0.SVID=T0$3.VID AND
T0$1.DVID=T0$4.VID AND
(T0$1.EL = n'submit' AND T0$1.EL IS NOT NULL) AND
(T0$3.T = 1 AND T0$3.V = n'employee') AND
(T0$2.T = 1 AND T0$2.V = n'employee') AND
(T0$4.T = 1 AND T0$4.V = n'order')
GROUP BY T0$2.T,
T0$2.V,
T0$2.VN,
T0$2.VT
ORDER BY (DECODE(T0$2.T,1,3,2,2,7,2,3,2,4,2,5,1,6,4)) ASC NULLS LAST,
T0$2.VN ASC NULLS LAST,
T0$2.VT ASC NULLS LAST,
```

```
DECODE(T0$2.T,6,DECODE(T0$2.V,n'Y',n'0',n'y',n'0',n'N',n'1',n'n',n'1'),T0
$2.V) ASC NULLS LAST
```

This is the generated SQL: a nice looking short PGQL query become a not-so-nice and not-so-short SQL query. But it's an automated process, not a single piece of SQL has been written by hand.