Relational DB vs Graph DB: And The Winner Is?

In 10 minutes not much is possible, therefore I will focus on a single use case, comparing an Oracle relational database with Oracle property graph.

Import packages and prepare environment

Connect to the database and PGX

In [1]:

```
from ipyleaflet import Map, Marker, AntPath, LayerGroup, Icon
import ipywidgets as widgets
from IPython.display import display
```

In [2]:

```
import cx_Oracle

dbConnection = cx_Oracle.connect(
    'my_user',
    'my_password',
    'db01.sub11131018510.datalysisvcn1.oraclevcn.com:1521/orclpdb1.sub11131018510.datal
ysisvcn1.oraclevcn.com'
)
print("Connectd to Oracle Database :", dbConnection.dsn)
dbCursor = dbConnection.cursor()
```

```
Connectd to Oracle Database : db01.sub11131018510.datalysisvcn1.oraclevcn.
com:1521/orclpdb1.sub11131018510.datalysisvcn1.oraclevcn.com
```

In [3]:

```
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/oracle/graph/lib'
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path="+_get_pgx_class_path(pgxLibsPa
th), convertStrings=True)

session = JClass('oracle.pgx.api.Pgx').createSession("http://localhost:7007/", "jupyter
Lab")
print("Connectd to PGX :", session.toString(), session.getServerInstance())
```

```
Connectd to PGX : PgxSession[ID=6e1e27a1-eccf-4397-bc7e-21f5b472f80a,sourc
e=jupyterLab] ServerInstance[embedded=false,baseUrl=http://localhost:700
7/,serverVersion=19.4.0]
```

Define locations and distances

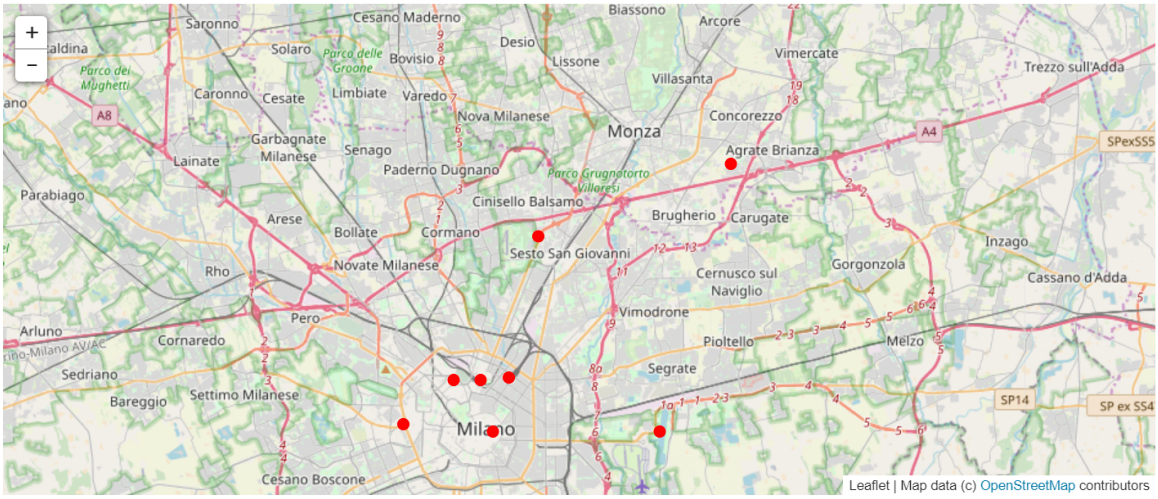
In [4]:

```
pois = {
    'A': {'label': 'Oracle Italia', 'lat': 45.5424524, 'lon': 9.219717},
    'B': {'label': 'Milano Centrale', 'lat': 45.4870123, 'lon': 9.2032903},
    'C': {'label': 'Google Italia', 'lat': 45.4861499, 'lon': 9.1873934},
    'D': {'label': 'Microsoft Italia', 'lat': 45.4861485, 'lon': 9.1720725},
    'E': {'label': 'Salesforce', 'lat': 45.4657091, 'lon': 9.1944653},
    'F': {'label': 'SAP Italia', 'lat': 45.57074, 'lon': 9.3276621},
    'G': {'label': 'Amazon Italia', 'lat': 45.4687648, 'lon': 9.1442233},
    'H': {'label': 'IBM Italia', 'lat': 45.4660765, 'lon': 9.2877713},
}

distances = [
    {'src': 'A', 'dst': 'B', 'distance': 16},
    {'src': 'A', 'dst': 'C', 'distance': 14},
    {'src': 'A', 'dst': 'D', 'distance': 20},
    {'src': 'A', 'dst': 'E', 'distance': 24},
    {'src': 'A', 'dst': 'F', 'distance': 16},
    {'src': 'A', 'dst': 'G', 'distance': 25},
    {'src': 'A', 'dst': 'H', 'distance': 22},
    {'src': 'B', 'dst': 'C', 'distance': 5},
    {'src': 'B', 'dst': 'D', 'distance': 6},
    {'src': 'B', 'dst': 'E', 'distance': 10},
    {'src': 'B', 'dst': 'F', 'distance': 28},
    {'src': 'B', 'dst': 'G', 'distance': 4},
    {'src': 'B', 'dst': 'H', 'distance': 28},
    {'src': 'C', 'dst': 'D', 'distance': 5},
    {'src': 'C', 'dst': 'E', 'distance': 12},
    {'src': 'C', 'dst': 'F', 'distance': 28},
    {'src': 'C', 'dst': 'G', 'distance': 4},
    {'src': 'C', 'dst': 'H', 'distance': 3},
    {'src': 'D', 'dst': 'E', 'distance': 12},
    {'src': 'D', 'dst': 'F', 'distance': 28},
    {'src': 'D', 'dst': 'G', 'distance': 4},
    {'src': 'D', 'dst': 'H', 'distance': 4},
    {'src': 'E', 'dst': 'F', 'distance': 28},
    {'src': 'E', 'dst': 'G', 'distance': 8},
    {'src': 'E', 'dst': 'H', 'distance': 10},
    {'src': 'F', 'dst': 'G', 'distance': 48},
    {'src': 'F', 'dst': 'H', 'distance': 28},
    {'src': 'G', 'dst': 'H', 'distance': 26},
]
```

Show map

In [5]:

```
# init map
mapCenter = (45.5369608,9.2427489) # Milano
mp = Map(center=mapCenter, zoom=11)

# display map
mp
```



Add locations on map

In [6]:

```
# layer group with the big markers
layer_bigMarkers = LayerGroup()
for k in pois:
    marker = Marker(
        location=(pois[k]['lat'], pois[k]['lon']),
        title=pois[k]['label'],
        draggable=False,
    )
    # add marker to layer group
    layer_bigMarkers.add_layer(marker)

# add layer group to map
mp.add_layer(layer_bigMarkers)
```

Switch to smaller markers

In [7]:

```
# remove the layer with big markers
mp.remove_layer(layer_bigMarkers)
```

In [8]:

```
# layer group with small markers
layer_markers = LayerGroup()
for k in pois:
    icon = Icon(
        icon_url='marker.png',
        icon_size=[10, 10],
        icon_anchor=[5,5],
        title=pois[k]['label'],
        alt=pois[k]['label'],
        draggable=False,
    )
    mark = Marker(location=(pois[k]['lat'], pois[k]['lon']), icon=icon)
    # add marker to layer group
    layer_markers.add_layer(mark)

# add layer to map
mp.add_layer(layer_markers)
```

Prepare database and graph

Create table and insert data

In [9]:

```
# dbConnection
# dbCursor

# https://blogs.oracle.com/oraclemagazine/perform-basic-crud-operations-using-cx-oracle
-part-1

# rows to insert into the DB table
rows = [(d['src'], d['dst'], d['distance']) for d in distances] + [(d['dst'], d['src'],
d['distance']) for d in distances]

# create table
statement = "CREATE TABLE connections (src varchar2(2) not null, dst varchar2(2) not nu
ll, distance number not null)"
dbCursor.execute(statement)

# insert rows
dbCursor.bindarraysize = 2
dbCursor.setinputsizes(1, 1, int)
dbCursor.executemany("INSERT INTO connections (src, dst, distance) values (:1, :2, :3)"
, rows)

# commit (and open new cursor for later usage)
dbConnection.commit()

print("Table 'connections' created and {} rows inserted".format(len(rows)))
```

```
Table 'connections' created and 56 rows inserted
```

Create graph

In [10]:

```
# create a graph builder
pgBuilder = session.createGraphBuilder(
    JClass('oracle.pgx.common.types.IdType').STRING,
    JClass('oracle.pgx.config.IdGenerationStrategy').USER_IDS,
    JClass('oracle.pgx.config.IdGenerationStrategy').AUTO_GENERATED,
)
# add edges (nodes created automatically)
for i in distances:
    pgBuilder.addEdge(i['src'], i['dst']).setProperty('distance', i['distance'])

# build graph
graph = pgBuilder.build().undirect('connections')
print(graph)
```

```
PgxGraph[name=connections,N=8,E=28,created=1581932415218]
```

Calculate shortest path

Between 'A' and 'G'

Relational database

In [11]:

```
# dbConnection
# dbCursor

# calculate the shortest path between 2 points
sql = """
WITH paths (
    src, dst, distance, lvl, way, totaldistance
) AS (
    SELECT src, dst, distance, 1 lvl, src || '-' || dst AS way, distance AS totaldistan
ce
    FROM connections
    WHERE src = 'A'
    UNION ALL
    SELECT c.src, c.dst, c.distance, p.lvl + 1, p.way || '-' || c.dst AS way, p.totaldi
stance + c.distance AS totaldistance
    FROM paths p
    JOIN connections c
    ON p.dst = c.src
) CYCLE src SET looped TO 'Y' DEFAULT 'N'
SELECT
way, totaldistance
FROM paths
WHERE dst = 'G'
AND looped = 'N'
AND INSTR(SUBSTR(way, 1, LENGTH(way)-1), 'G') = 0
ORDER BY totaldistance ASC
FETCH FIRST 5 ROWS ONLY
"""
print(sql)
print('')

dbCursor.execute(sql)
res = dbCursor.fetchall()

# iterate over results
for r in res:
    #print("path {} has a total distance of {}".format(r[0], r[1]))
     print("From '{}' to '{}' it takes {} via {}".format(
        'A',
        'G',
        r[1],
        r[0]
    ))
```

```
WITH paths (
    src, dst, distance, lvl, way, totaldistance
) AS (
    SELECT src, dst, distance, 1 lvl, src || '-' || dst AS way, distance A
S totaldistance
    FROM connections
    WHERE src = 'A'
    UNION ALL
    SELECT c.src, c.dst, c.distance, p.lvl + 1, p.way || '-' || c.dst AS w
ay, p.totaldistance + c.distance AS totaldistance
    FROM paths p
    JOIN connections c
    ON p.dst = c.src
) CYCLE src SET looped TO 'Y' DEFAULT 'N'
SELECT
way, totaldistance
FROM paths
WHERE dst = 'G'
AND looped = 'N'
AND INSTR(SUBSTR(way, 1, LENGTH(way)-1), 'G') = 0
ORDER BY totaldistance ASC
FETCH FIRST 5 ROWS ONLY
```

```
From 'A' to 'G' it takes 18 via A-C-G
From 'A' to 'G' it takes 20 via A-B-G
From 'A' to 'G' it takes 23 via A-C-D-G
From 'A' to 'G' it takes 23 via A-C-B-G
From 'A' to 'G' it takes 24 via A-D-G
```

Property Graph

In [12]:

```
pgql = """
SELECT ID(start) AS start, SUM(e.distance) AS distance, ID(dst) AS dst, ARRAY_AGG( ID(s
tep) ) AS path
MATCH TOP 50 SHORTEST( (start) (-[e]-> (step))* (dst) )
WHERE ID(start) = 'A' AND ID(dst) = 'G'
AND COUNT(DISTINCT e) = COUNT(e)
ORDER BY distance ASC
LIMIT 5
"""
print(pgql)
print('')

# execute query
pgxResultSet = graph.queryPgql(pgql)
print("PGQL query has {} result(s)".format(pgxResultSet.getNumResults()))
print('')

# iterate over results
pgxResults = pgxResultSet.getResults()
for r in pgxResults.iterator():
    # get path (need to append the start point)
    path = list(r.getList('path'))
    path.insert(0, r.getString('start'))

    print("From '{}' to '{}' it takes {} via {}".format(
        r.getString('start'),
        r.getString('dst'),
        r.getLong('distance'),
        '-'.join(path)
    ))

# destroy result object
pgxResultSet.close()
```

```
SELECT ID(start) AS start, SUM(e.distance) AS distance, ID(dst) AS dst, AR
RAY_AGG( ID(step) ) AS path
MATCH TOP 50 SHORTEST( (start) (-[e]-> (step))* (dst) )
WHERE ID(start) = 'A' AND ID(dst) = 'G'
AND COUNT(DISTINCT e) = COUNT(e)
ORDER BY distance ASC
LIMIT 5


PGQL query has 5 result(s)

From 'A' to 'G' it takes 18 via A-C-G
From 'A' to 'G' it takes 20 via A-B-G
From 'A' to 'G' it takes 23 via A-C-B-G
From 'A' to 'G' it takes 23 via A-C-D-G
From 'A' to 'G' it takes 24 via A-D-G
```

Calculate shortest path

(test paths: *A-G*, *G-F*, *A-D*)

In [13]:

```
# capture widgets output
output = widgets.Output()

# define widgets
srcRadioButtons = widgets.RadioButtons(
    options=list(pois.keys()),
    description='Start point:',
    disabled=False
)
dstRadioButtons = widgets.RadioButtons(
    options=list(pois.keys()),
    description='End point:',
    disabled=False
)

findPathButton = widgets.Button(
    description='Find path',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Find shortest path between points',
)

ctrlBox = widgets.HBox([srcRadioButtons, dstRadioButtons, findPathButton])

fullWidget = widgets.VBox([ctrlBox, mp])

# create src and dst markers and place them
specialIcon = Icon(
    icon_url='http://140.238.216.196:8888/files/GitHub/jupyterlab/Rel%20vs%20Graph%20D
B%2010min/marker2.png?_xsrf=2%7C0434f9e5%7C53dded48f36dda0d9354af0744b7e10a%7C157970194
7',
    icon_size=[10, 10],
    icon_anchor=[5,5],
    draggable=False,
)
layer_pathMarkers = LayerGroup()
srcMark = Marker(location=(0,0), icon=specialIcon)
layer_pathMarkers.add_layer(srcMark)
dstMark = Marker(location=(0,0), icon=specialIcon)
layer_pathMarkers.add_layer(dstMark)
# path connecting src and dst
antPathSrc2Dst = AntPath(
    locations = [[0,0], [0,0]],
    dash_array = [1, 10],
    delay = 1000,
    color = '#7590ba',
    pulse_color = '#3f6fba'
)
layer_pathMarkers.add_layer(antPathSrc2Dst)
# add the layer group to map
mp.add_layer(layer_pathMarkers)


# display widget
display(fullWidget, output)

### events handlers ###
# change src or dst value => move marker
def on_srcRadio_change(change) :
```

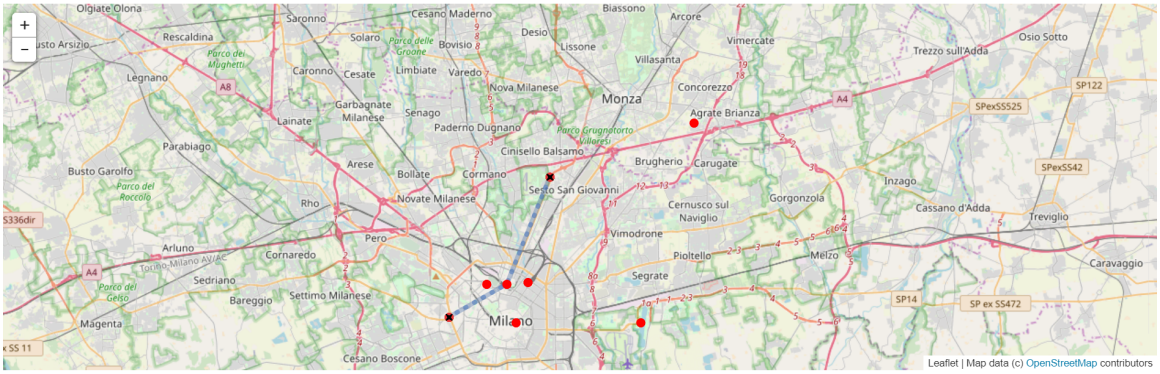
```
    srcMark.location=(pois[change['new']]['lat'], pois[change['new']]['lon'])
def on_dstRadio_change(change) :
    dstMark.location=(pois[change['new']]['lat'], pois[change['new']]['lon'])
# button clicked
def on_findPathButton_click(b) :
    # disable button
    findPathButton.disabled = True
    # do something
    output.clear_output()
    with output:
        print("Finding shortest path from '{}' to '{}' ...".format(srcRadioButtons.valu
e, dstRadioButtons.value))
        print("")
        print(pgx_find_path())
        print(db_find_path())
    # re-enable button
    findPathButton.disabled = False

def db_find_path() :
    msg = ""
    sql = """
    WITH paths (
        src, dst, distance, lvl, way, totaldistance
    ) AS (
        SELECT src, dst, distance, 1 lvl, src || '-' || dst AS way, distance AS totaldi
stance
        FROM connections
        WHERE src = '{}'
        UNION ALL
        SELECT c.src, c.dst, c.distance, p.lvl + 1, p.way || '-' || c.dst AS way, p.tot
aldistance + c.distance AS totaldistance
        FROM paths p
        JOIN connections c
        ON p.dst = c.src
    ) CYCLE src SET looped TO 'Y' DEFAULT 'N'
    SELECT
    way, totaldistance
    FROM paths
    WHERE dst = '{}'
    AND looped = 'N'
    AND INSTR(SUBSTR(way, 1, LENGTH(way)-1), '{}') = 0
    ORDER BY totaldistance ASC
    FETCH FIRST 1 ROWS ONLY
    """.format(srcRadioButtons.value, dstRadioButtons.value, dstRadioButtons.value)
    dbCursor.execute(sql)
    res = dbCursor.fetchall()

    # iterate over results
    for r in res:
        msg = "[DB]  From '{}' to '{}' it takes {} minutes via {}".format(
            srcRadioButtons.value,
            dstRadioButtons.value,
            r[1],
            r[0]
        )

    # return msg
    return msg

def pgx_find_path() :
    msg = ""
```

```
    pgql = """
    SELECT ID(start) AS start, SUM(e.distance) AS distance, ID(dst) AS dst
    , ARRAY_AGG( ID(step) ) AS path
    MATCH TOP 50 SHORTEST( (start) (-[e]-> (step))* (dst) )
    WHERE ID(start) = '{}' AND ID(dst) = '{}' AND COUNT(DISTINCT e) = COUNT(e)
    ORDER BY distance ASC
    LIMIT 1
    """.format(srcRadioButtons.value, dstRadioButtons.value)

    # execute query
    pgxResultSet = graph.queryPgql(pgql)

    # iterate over results
    pgxResults = pgxResultSet.getResults()
    path = []
    for r in pgxResults.iterator():
        # get path (need to append the start point)
        path = list(r.getList('path'))
        path.insert(0, r.getString('start'))

        msg = "[PGX] From '{}' to '{}' it takes {} minutes via {}".format(
            r.getString('start'),
            r.getString('dst'),
            r.getLong('distance'),
            '-'.join(path)
        )

    # destroy result object
    pgxResultSet.close()

    # display ant path
    locations = []
    for k in path:
        locations.append([pois[k]['lat'], pois[k]['lon']])
    position_ant_path(locations)

    # return msg
    return msg

def position_ant_path(locations) :
    antPathSrc2Dst.locations = locations

# attach handlers
srcRadioButtons.observe(on_srcRadio_change, names='value')
dstRadioButtons.observe(on_dstRadio_change, names='value')
findPathButton.on_click(on_findPathButton_click)

# reposition src and dst markers
on_srcRadio_change({'new': srcRadioButtons.value})
on_dstRadio_change({'new': dstRadioButtons.value})
```

Start point: A B C D E F G H

End point: A B C D E F G H

Find path




```
Finding shortest path from 'A' to 'G' ...

[PGX] From 'A' to 'G' it takes 18 minutes via A-C-G
[DB]  From 'A' to 'G' it takes 18 minutes via A-C-G
```

In [14]:

```
# clean-up map
mp.remove_layer(layer_pathMarkers)
```

Clean-up

Relational database

Drop table and close connection

In [ ]:

```
# dbConnection
# dbCursor

# drop table
statement = "DROP TABLE connections"
dbCursor.execute(statement)

# close cursor
dbCursor.close()

# close connection
dbConnection.close()
```

Property Graph

Drop graph and close session

In [ ]:

```
# drop graph
graph.destroy()

# close session
session.close()
```

In [ ]:

```

```