Graph Analytics Using the Python API

Melli Annamalai and Ryota Yamanaka, Product Management, Oracle
October 29, 2020
AskTOM Office Hours: Graph Database and Analytics

• Welcome to our AskTOM Graph Office Hours series! We’re back with new product updates, use cases, demos and technical tips [https://asktom.oracle.com/pls/apex/asktom.search?oh=3084](https://asktom.oracle.com/pls/apex/asktom.search?oh=3084)

• Sessions will be held about once a month

• **Subscribe** at the page above for updates on upcoming session topics & dates And submit feedback, questions, topic requests, and view past session recordings

• **Note:** **Spatial** now has a new Office Hours series for location analysis & mapping features in Oracle Database: [https://asktom.oracle.com/pls/apex/asktom.search?oh=7761](https://asktom.oracle.com/pls/apex/asktom.search?oh=7761)
Agenda

1. Recap – Graph Analytics and APIs
2. Basic Operations of Python API
3. Demo: Convert and Load Data from Database
4. Demo: Query, Run Algorithms, and Visualize
5. Demo: Combine with Machine Learning
Recap – Graph Analytics and APIs
Graph Database and Analytics

Store, manage, query, and analyze graphs
  • Enterprise capabilities: Built on Oracle infrastructure
  • Manageability, fine-grained security, high availability, integration

Highly scalable
  • In-memory query and analytics and in-database query
  • 10s of billions of edges and vertices

PGQL: Powerful SQL-like graph query language

Analytics: 50+ pre-built graph analysis algorithms
  Java and Python APIs

Visualization
  • Light-weight web application, UI accessible from a browser
Graph APIs and Clients

- **Java API** for PGQL queries and graph analytics
  
  
  ```
  opg-jshell> session.queryPgql("SELECT e from MATCH ()-[e]->()")
  opg-jshell> analyst.pagerank(my_graph)
  ```

- **Zeppelin notebook (PGX interpreters), Java application**

- **New in Oracle Graph Server and Client 20.4**

- **Python API** for PGQL queries and graph analytics in the in-memory graph server (PGX)

- **SQLcl** for PGQL queries (20.3 onward)

  ```
  SQL> pgql auto on
  PGQL> SELECT e from MATCH ()-[e]->();
  ```
Why Python?

Enable data scientists to easily work with graphs

Integration with data science tools and environments
Python Client

- Python module is called `pypgx`
- Can be used
  - Interactively (Python shell), or
  - As module imported into a Python application

- Works with graphs in PGX

- If using PGQL-in-database
  - Use PGQL in SQLcl to create graph and run queries
  - Use Python API to load into PGX for analytics
Interactive Shells

• Graph Client includes shells for Java and Python

• To use Java API interactively, run opg-jshell:

  $ ./bin/opg-jshell -b http://graph-server:7007 --username graph_dev
  enter password for user graph_dev (press Enter for no password):
opg-jshell>

• To use Python API interactively, run opgpy:

  $ ./bin/opgpy -b http://graph-server:7007 --username graph_dev
  enter password for user graph_dev (press Enter for no password):
  >>>
Zeppelin Notebook

The interpreter for Apache Zeppelin Notebook is provided and **Groovy** syntax is supported.

Users can run PGQL queries and graph algorithms.
No additional interpreter is needed. But **pypgx** should be installed on the notebook server.

Jupyter is a popular open-source notebook interface (under modified BSD license, formally called IPython)
Basic Operations of Python API
Installation of Python API

- Python version 3.5 or later is required

$ python3 --version
Python 3.6.1

- JDK 8 or later is required

$ java --version
java 11.0.8 2020-07-14 LTS
Installation of Python API


• Install the required dependencies

$ pip3 install Cython six pyjnius

• Install the client

$ pip3 install oracle-graph-client-20.4.0.zip
Connect to Graph Server

• Connect to Graph Server using the interactive shell
• Authenticate as a database

$ ./bin/opgpy -b https://graph-server:7007 --username graph_dev
Create Graph using DDL

• Load data from database tables and create graph in graph server

```python
>>> statement = 'CREATE PROPERTY GRAPH "My Graph" ...
>>> session.prepare_pgql(statement).execute()
```
Get Graph on Graph Server

• Use graphs already loaded on Graph Server

```python
>>> my_graph = session.get_graph("My Graph")
```
Run PGQL Queries

- Run Query against the graph and retrieve the result as result_set

```python
>>> my_graph.query_pgql("SELECT m.name FROM MATCH (n)-[:likes]-(m) ...")
```
Run Graph Algorithms

- Run Query against the graph and update the graph

```python
>>> analyst = session.create_analyst()
>>> result = analyst.pagerank(my_graph);
```
Demo: Convert and Load Data from Database
Setup Quickstart Environment

Installation: https://github.com/ryotayamanaka/oracle-pg/tree/20.4

Clone repository

(Note, the branch is 20.4)

$ git clone https://github.com/ryotayamanaka/oracle-pg.git -b 20.4

Download and extract packages

(Note, the packages are version 20.4)

$ sh extract.sh

Build and start the docker containers

$ docker-compose up
Example - Product Purchase Dataset

- Online Retail dataset (Kaggle)
  - 4,339 customers
  - 3,919 products
  - 396,370 purchases

Example - Product Recommendation in Online Retail

When you need to analyze the product purchase activity of your customers and generate recommendation, your graph should contain the following information:

- **Customer** entities
- **Product** entities
- **Purchased** relationships
Convert and Load Data from Database

- Load data from database tables and create graph in graph server

```python
>>> statement = 'CREATE PROPERTY GRAPH "Online Retail" ...'
>>> session.prepare_pgql(statement).execute()
```
CREATE PROPERTY GRAPH "Online Retail"

VERTEX TABLES (  
  online_retail.customers  
  LABEL "Customer"  
  PROPERTIES (customer_id AS "customer_id", "country")  
  , online_retail.products  
  LABEL "Product"  
  PROPERTIES (stock_code AS "stock_code", "description")  
)

EDGE TABLES (  
  ...  
)
CREATE PROPERTY GRAPH Statement

• Definition of edges

CREATE PROPERTY GRAPH "Online Retail"
  VERTEX TABLES (  
    ...  
  )  
EDGE TABLES (  
  online_retail.purchases_distinct  
    KEY (purchase_id)  
    SOURCE KEY(customer_id) REFERENCES customers  
    DESTINATION KEY(stock_code) REFERENCES products  
    LABEL "has_purchased"  
    PROPERTIES (purchase_id)  
  )
Demo: Query, Run Algorithms, and Visualize
Get Graph on Graph Server

- Attach the graphs already loaded on Graph Server

```python
>>> graph = session.get_graph("Online Retail")
```
Run Algorithm

For calculating the recommended products for a specific customer "cust_12353", we use **personalize pagerank** algorithm here.

Firstly, get the vertex object represents the customer.

```python
rs = graph.query_pgql(
    "SELECT ID(c) FROM MATCH (c) WHERE c.customer_id = 'cust_12353'"
vertex = graph.get_vertex(rs.get_row(0))
```

Run the algorithm. The result is stored as "ppr" new node property.

```python
analyst.personalized_pagerank(graph, vertex, rank="ppr")
```
Query the Algorithm Results

After running algorithms, the results are stored into the graph, e.g. each node gets a new property called "ppr".

Users can access the results using PGQL queries:

```
SELECT p.description, p.ppr
FROM MATCH (p)
ORDER BY p.ppr DESC
```
Visualize the Results

Login with the same **session ID** as the one which ran the algorithms.

The size of nodes can be linked to the pagerank scores.
Demo: Combine with Machine Learning
How Graph Enhances Machine Learning

- **Information coverage** of the training dataset is important to make good predictive models
- New features are generated based on relationships using **graph queries** and **graph algorithms**

<table>
<thead>
<tr>
<th>ID</th>
<th>Feature1</th>
<th>Feature 2</th>
<th>Feature 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data source ➔ ML algorithms ➔ Prediction model

<table>
<thead>
<tr>
<th>Target</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
How Graph Enhances Machine Learning

- **Information coverage** of the training dataset is important to make good predictive models
- New features are generated based on relationships using **graph queries** and **graph algorithms**

<table>
<thead>
<tr>
<th>ID</th>
<th>Feature1</th>
<th>Feature 2</th>
<th>Feature 3</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>1002</td>
<td></td>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>1003</td>
<td></td>
<td></td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
How Graph Enhances Machine Learning

- **Information coverage** of the training dataset is important to make good predictive models.
- New features are generated based on relationships using **graph queries** and **graph algorithms**.
Example - Mule Account Detection

- **Mule accounts** are often stolen accounts and transfers money illegally

- Question:
  Is it possible to generate **more features** based on the relationships between accounts (e.g. transaction patterns, family relationships, ...)?
Example - Mule Account Detection

Feature 1

• If the owner of this account is sharing personal information with others

```sql
SELECT a1, COUNT(s)
MATCH (a1)<-[:owns]-(c1)-[:owns]-(c2)-[:owns]>(a2)
WHERE a1 != a2 AND c1 != c2
GROUP BY a1
```
Example - Mule Account Detection

Feature 2

• How many fraud accounts exist in the same money transfer community

```sql
SELECT a.scc_kosaraju, COUNT(a)
MATCH (a)
WHERE a.type = 'Account'
  AND a.is_fraud = 'true'
GROUP BY a.scc_kosaraju
```
Example - Mule Account Detection

Feature 3

• Closeness to know fraud accounts

```sql
SELECT a.account_no, a.pagerank, a.is_fraud
MATCH (a) WHERE a.type = 'Account'
ORDER BY a.pagerank DESC
```

```
analyst.personalized_pagerank(G, fraud_accounts)
```
Example - Mule Account Detection

Database

Account

name
age
occupation
branch
balance
...
share_info
fraud_community
fraud_closeness
...
is_fraud
### Example - Mule Account Detection

**Data source**

<table>
<thead>
<tr>
<th>ID</th>
<th>age</th>
<th>occupation</th>
<th>balance</th>
<th>share_info</th>
<th>fraud_com</th>
<th>fraud_clo</th>
<th>is_fraud</th>
<th>prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Python**

**Graph view**

**Prediction model**

- Yes 85%
- Yes 78%
- No 62%
Helpful Links

• Oracle Property Graph: [http://www.oracle.com/goto/propertygraph](http://www.oracle.com/goto/propertygraph)

• Hands-on (using Oracle Cloud)

• Social Media
  • Twitter: @OracleBigData, @SpatialHannes, @JeanIhm, @ryotaymnk, @AnnamalaiMelli
  • LinkedIn: Oracle Spatial and Graph Group
  • YouTube: [youtube.com/c/OracleSpatialandGraph](https://youtube.com/c/OracleSpatialandGraph)
AskTOM Office Hours:  Graph Database and Analytics

• Welcome to our AskTOM Graph Office Hours series! We’re back with new product updates, use cases, demos and technical tips. [Link](https://asktom.oracle.com/pls/apex/asktom.search?oh=3084)

• Sessions will be held about once a month

• **Subscribe** at the page above for updates on upcoming session topics & dates. And submit feedback, questions, topic requests, and view past session recordings.

• **Note:** **Spatial** now has a new Office Hours series for location analysis & mapping features in Oracle Database: [Link](https://asktom.oracle.com/pls/apex/asktom.search?oh=7761)
Our mission is to help people see data in new ways, discover insights, unlock endless possibilities.