Unlocking graph analytics in DuckDB with SQL/PGQ

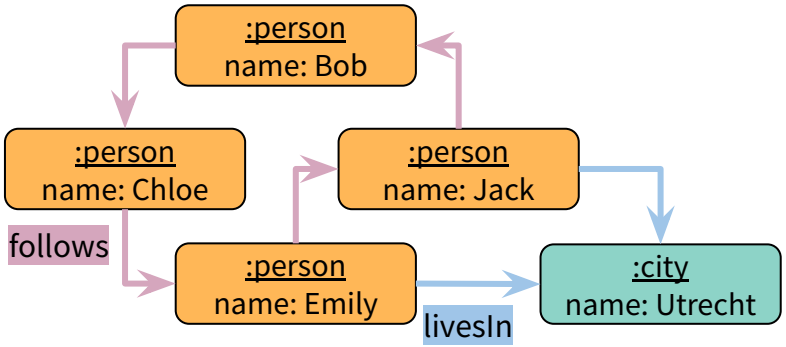
Daniël ten Wolde
Ph.D. student
CWI Database Architectures group

Storing graphs in SQL

```
CREATE TABLE city (
  id bigint PRIMARY KEY,
  name varchar
);

CREATE TABLE person (
  id bigint PRIMARY KEY,
  name varchar
);
CREATE TABLE livesIn (
  personid bigint,
  cityid bigint
);

CREATE TABLE follows (
  p1id bigint,
  p2id bigint
);
```



Prompt: Count the number of people Bob (in)directly follows who live in the city Utrecht

Prompt: Count the number of people Bob (in)directly follows who live in the city Utrecht

SQL:1999 query

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode, array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN livesIn l on p2.id = l.personid
JOIN city c     ON c.id = l.cityid AND c.name = 'Utrecht';
```

Prompt: Count the number of people Bob (in)directly follows who live in the city Utrecht

SQL:1999 query

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode, array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN livesIn l on p2.id = l.personid
JOIN city c     ON c.id = l.cityid AND c.name = 'Utrecht';
```

Prompt: Count the number of people Bob
(in)directly follows who live in the city Utrecht

SQL:1999 query

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode, array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN livesIn l on p2.id = l.personid
JOIN city c     ON c.id = l.cityid AND c.name = 'Utrecht';
```

SQL/PGQ (Property Graph Queries)

- Part of SQL:2023 standard
- Property graph layer over existing tables
- Visual graph syntax
  - Pattern matching
  - Path-finding

SQL/PGQ property graph creation

```
CREATE PROPERTY GRAPH SocialNetwork
  VERTEX TABLES (
    person, city
  )
  EDGE TABLES (
    follows    SOURCE KEY (p1id) REFERENCES person (id)
               DESTINATION KEY (p2id) REFERENCES person (id),
    livesIn    SOURCE KEY (personid) REFERENCES person (id)
               DESTINATION KEY (cityid) REFERENCES city (id)
);
```

SQL/PGQ query

Prompt: Count the number of people Bob
(in)directly follows who live in the city Utrecht

```
SELECT count(id)
FROM
  GRAPH_TABLE (SocialNetwork
    MATCH p = ANY SHORTEST (p1:person WHERE p1.name='Bob')
        -[f:follows]->*(p2:person)
        -[l:livesIn]->(c:city WHERE c.name='Utrecht')
    COLUMNS (p2.id));
```

SQL/PGQ query

Prompt: Count the number of people Bob (in)directly follows who live in the city Utrecht

```
SELECT count(id)
FROM
  GRAPH_TABLE (SocialNetwork
    MATCH p = ANY SHORTEST (p1:person WHERE p1.name='Bob')
        -[f:follows]->*(p2:person)
        -[l:livesIn]->(c:city WHERE c.name='Utrecht')
    COLUMNS (p2.id));
```

SQL/PGQ query

Prompt: Count the number of people Bob (in)directly follows who live in the city Utrecht

```
SELECT count(id)
FROM
  GRAPH_TABLE (SocialNetwork
    MATCH p = ANY SHORTEST (p1:person WHERE p1.name='Bob')
        -[f:follows]->*(p2:person)
        -[l:livesIn]->(c:city WHERE c.name='Utrecht')
    COLUMNS (p2.id));
```

SQL/PGQ query

Prompt: Count the number of people Bob (in)directly follows who live in the city Utrecht

```
SELECT count(id)
FROM
  GRAPH_TABLE (SocialNetwork
    MATCH p = ANY SHORTEST (p1:person WHERE p1.name='Bob')
        -[f:follows]->*(p2:person)
        -[l:livesIn]->(c:city WHERE c.name='Utrecht')
    COLUMNS (p2.id));
```

SQL/PGQ query

Prompt: Count the number of people Bob
(in)directly follows who live in the city Utrecht

```
SELECT count(id)
FROM
  GRAPH_TABLE (SocialNetwork
    MATCH p = ANY SHORTEST (p1:person WHERE p1.name='Bob')
        -[f:follows]->*(p2:person)
        -[l:livesIn]->(c:city WHERE c.name='Utrecht')
    COLUMNS (p2.id));
```

SQL/PGQ

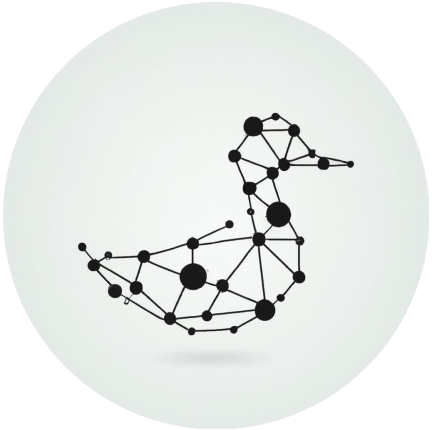
```
SELECT count(id) AS cp2
FROM GRAPH_TABLE (socialNetwork
  MATCH p = ANY SHORTEST (p1:person WHERE p1.name='Bob')-[:follows]->*(p2:person)
          -[:livesIn]->(c:city WHERE c.name='Utrecht')
  COLUMNS (p2.id));
```

plain SQL

```
WITH RECURSIVE paths(startNode, endNode, path) AS (
   SELECT p1id AS startNode, p2id AS endNode, ARRAY[p1id, p2id] AS path
     FROM follows JOIN person p1 ON p1.id = follows.p1id WHERE p1.name = 'Bob'
   UNION ALL (
     WITH paths AS (TABLE paths)
        SELECT paths.startNode AS startNode, p2id AS endNode,
           array_append(path, p2id) AS path
        FROM paths JOIN follows ON paths.endNode = follows.p1id
        WHERE NOT EXISTS (SELECT true FROM paths previous_paths
                          JOIN person p2 ON p2.id = follows.p2id
                          WHERE p2.name = 'Bob' OR follows.p2id = previous_paths.endNode)))
SELECT count(p2.id) AS cp2
FROM person p1
JOIN paths     ON paths.startNode = p1.id
JOIN person p2 ON p2.id = paths.endNode
JOIN livesIn l on p2.id = l.personid
JOIN city c     ON c.id = l.cityid AND c.name = 'Utrecht';
```

The SQL/PGQ query is 4× shorter & more readable

DuckPGQ extension

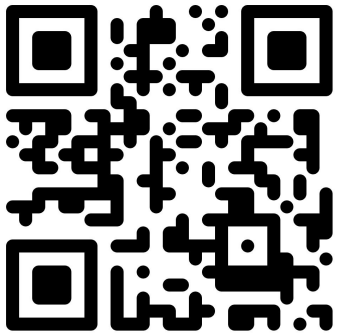


- Installable as a community extension
- Translated to standard relational query plans
- Special UDFs for path-finding
- Graph algorithms as table functions:
    a. PageRank
    b. Weakly Connected Component
    c. Local Clustering Coefficient

Try DuckPGQ out in DuckDB

```
> duckdb
● install duckpgq from community;
● load duckpgq;
```

duckpgq.org



dtenwolde.bsky.social dljtw@cwi.nl