DuckPGQ: Efficient Property Graph Queries in an analytical RDBMS

Daniël ten Wolde, Tavneet Singh, Gabor Szarnyas, Peter Boncz
CWI Database Architectures group

CIDR 2023
Amsterdam
Outline

1. the why and what of SQL/PQG

2. competent graph database systems architecture

3. graph query processing in DuckDB
Graph data management

connected data

tables often represent graphs
Graph data management

connected data

This image illustrates the concept of graph data management with a visual representation of connected data. Tables often represent graphs, with examples of relational operators shown in the form of tables:

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Important functionalities:

- **Pattern matching**
- **Path-finding**
- **Relational operators**

The SQL query shown is:

```sql
SELECT count(*)
FROM person
WHERE name LIKE 'E%'
```
Storing graphs in SQL

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

CREATE TABLE person (
    id bigint PRIMARY KEY,
    name varchar,
    livesIn bigint,
    CONSTRAINT c FOREIGN KEY (livesIn) REFERENCES city (id)
);

CREATE TABLE follows (p1id bigint,
    p2id bigint,
    CONSTRAINT p1 FOREIGN KEY (p1id) REFERENCES person (id),
    CONSTRAINT p2 FOREIGN KEY (p2id) REFERENCES person (id) );
“count the number of people Bob (in)directly follows who live in the city Utrecht”
“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 city ON city.id = p2.livesIn AND city.name = 'Utrecht'
"count the number of people Bob (in)directly follows who live in the city Utrecht”
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 city ON city.id = p2.livesIn AND city.name = 'Utrecht'
Graph query languages

- **NebulaGraph**: nGQL
- **Amazon Neptune**: SPARQL
- **JanusGraph**: Gremlin
- **Oracle Labs PGX**: PGQL
- **TigerGraph**: GSQL
- **neo4j**: Cypher
The (sorry) State of Graph Database Systems

Peter Boncz
CWI

comparing graph with relational database systems...
+ provide pointers to related literature
The Sorry State of Graph Database Systems

“The six blunders of graph database systems” (see keynote)

- time may be running out for native property graph database systems
  - Some success in certain use cases: Data Integration, Data cleaning & Enrichment, Fraud Detection, Recommendation, Historical Analysis, Root-Cause Analysis,...
  - still a niche solution and maturity+usability problems remain

- especially if SQL/PGQ becomes a (moderate) success
  - Relational systems will be able to handle their use cases
  - Only Data Integration, Data cleaning & Enrichment would be left (RDF/SPARQL territory)
SQL/PGQ (Property Graph Queries)
SQL/PGQ

- Extension in the upcoming SQL:2023 standard, 2b released in June
- Property Graphs as views over existing tables
  - edge, vertex = table, property (value) = column (value), label = table-name
- Read-only operations for property graph queries
  - Path-finding + Pattern matching in Cypher-like syntax, producing a “Graph-Table” in FROM
SQL/PGQ

- Extension in the upcoming SQL:2023 standard, 2b released in June
- Property Graphs as views over existing tables
  - edge, vertex = table, property (value) = column (value), label = table-name
- Read-only operations for property graph queries
  - Path-finding + Pattern matching in Cypher-like syntax, producing a “Graph-Table” in FROM

TigerGraph
GSQl
Oracle Labs PGX
PGQL
neo4j
Cypher
LDBC
G-CORE
ISO
SQL:2023
Tabular schema

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

CREATE TABLE person (  
    id bigint PRIMARY KEY,  
    name varchar,  
    livesIn bigint,  
    CONSTRAINT c FOREIGN KEY ...  
);

CREATE TABLE follows (  
    p1id bigint,  
    p2id bigint,  
    CONSTRAINT p1 FOREIGN KEY ...  
    CONSTRAINT p2 FOREIGN KEY ...  
);

SQL/PGQ graph tables

CREATE PROPERTY GRAPH socialNetwork  
VERTEX TABLES (  
    city,  
    person  
)

EDGE TABLES (  
    livesIn SOURCE person DESTINATION city,  
    follows SOURCE person DESTINATION person  
);
SQL/PGQ query

“count the number of people Bob (in)directly follows who live in the city Utrecht”

SELECT count(gt.id)
FROM
  GRAPH_TABLE (socialNetwork,)
  MATCH (p1:person WHERE p1.name='Bob')-[[:follows]->* (p2:person)
    -[:livesIn]-(c:city WHERE c.name='Utrecht')
  ) gt
  COLUMNS (p2.id)
DuckPGQ module for DuckDB
DuckDB

- open-source in-process SQL OLAP DBMS
- Created by Mark Raasveldt & Hannes Mühleisen (keynote Wednesday)
- very popular in data science notebooks, but suitable for many analytics applications
- “Modern”: Vectorized execution engine, Morsel-driven parallelism,..
- Allows extension modules:
  - scalar user-defined functions (UDF), parser extensions
  - data sources (scans), table-returning functions
Current DuckPGQ pipeline

SQL/PQG graph creation query

register property graph views on vertex and edge tables

Database catalog

t_person

t_city

t_follows

t_livesIn

person
city
follows
livesIn

Base tables   Graph view
Current DuckPGQ pipeline

- SQL/PGQ graph creation query
- SQL/PGQ query
- Register property graph views on vertex and edge tables

Database catalog

SQL query

UDFs

Base tables

Graph view

- t_person
- t_city
- t_follows
- t_livesIn

- person
- city
- follows
- livesIn
Path finding: Compressed Sparse Row (CSR)

- **On-the-fly** creation (no update handling needed)
- Using **scalar UDFs** (parallel, very fast)
- Index in the **vertex array** corresponds to the ROWID of the vertex
- Vertex array contains offsets for the **edge arrays**
Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)
Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

**Initial State**

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Visit**

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Seen**

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Multi-Source Breadth-First Search (MS-BFS)**

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)
Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

VLDB’14

The More the Merrier: Efficient Multi-Source Graph Traversal

Manuel Then†
then@in.tum.de
Ken Pham‡
pham@nyu.edu
Mayritz Kaufmann* kaufmann@in.tum.de
Tuan-Anh Hoang-Vu‡
huyvu@nyu.edu
Alfons Kemper*
kemper@in.tum.de
Fernando Chirigati†
chirigati@nyu.edu
Thomas Neumann* neumann@in.tum.de
Huy T. Vo†
huy.vo@nyu.edu

* Technische Universität München
† New York University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Most graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose Multi-Source BFS (MS-BFS), an algorithm that is designed to

have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [20].

In a wide range of graph analytics algorithms, including shortest path computation [19], graph centrality calculation [9, 25], and k-hop neighborhood detection [12], breadth-first search (BFS)-based graph traversal is an essential building block used to systematically traverse a graph, i.e., to visit all reachable vertices and edges of the graph from a
Multi-Source Breadth-First Search (MS-BFS)

- Batched variant developed by Manuel Then
  - Works like regular BFS, but starts from multiple nodes
- Share the memory access
  - Major bottleneck
  - Can make use of SIMD instructions (SSE/AVX)

VLDB’14

The More the Merrier: Efficient Multi-Source Graph Traversal

MANUEL THEN\textsuperscript{1} KIEN PHAM\textsuperscript{2} ALFONS KEMPER\textsuperscript{1} MANUEL THEN\textsuperscript{1} MYRITZ KAUFMANN\textsuperscript{2} FERNANDO CHIRIGATI\textsuperscript{3} TUAN-ANH HOANG-VU\textsuperscript{1} THOMAS NEUMAN\textsuperscript{2} HUY T. VO\textsuperscript{1}

\textsuperscript{1}Technische Universität München\textsuperscript{2}IN TUM\textsuperscript{3}Columbia University

ABSTRACT

Graph analytics on social networks, Web data, and communication networks has been widely used in a plethora of applications. Many graph analytics algorithms are based on breadth-first search (BFS) graph traversal, which is not only time-consuming for large datasets but also involves much redundant computation when executed multiple times from different start vertices. In this paper, we propose Multi-Source BFS (MS-BFS), an algorithm that is designed to have influence on others and, as a consequence, are of great importance to spread information, e.g., for marketing purposes [28].

In a wide range of graph analytics algorithms, including shortest path computation [19], graph centrality calculation [9, 27], and k-hop neighborhood detection [12], breadth-first search (BFS)-based graph traversal is an elementary building block used to systematically traverse a graph, i.e., to visit all reachable vertices and edges of the graph from a given start vertex. However, the effectiveness of BFS in solving these problems depends on the ability to efficiently traverse the graph. In this paper, we propose Multi-Source BFS (MS-BFS), an algorithm that is designed to efficiently traverse a graph from multiple start vertices.

BFS 2nd level

\begin{tabular}{|c|c|}
\hline
B1 & B2 \\
\hline
1 & \textbf{X} \\
2 & \textbf{X} \\
3 & \\
4 & \\
5 & \textbf{X} \textbf{X} \\
6 & \textbf{X} \textbf{X} \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
B1 & B2 \\
\hline
1 & \textbf{X} \textbf{X} \\
2 & \textbf{X} \textbf{X} \\
3 & \textbf{X} \textbf{X} \\
4 & \textbf{X} \textbf{X} \\
5 & \textbf{X} \textbf{X} \\
6 & \textbf{X} \textbf{X} \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
1 & \\
2 & \\
3 & \\
4 & \\
5 & \\
6 & \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
1 & \\
2 & \\
3 & \\
4 & \\
5 & \\
6 & \\
\hline
\end{tabular}

Visit

Seen
Conclusion

- Why should you read our DuckPGQ paper?
  - Learn SQL/PGQ in less than 1 page (or become ldbcouncil.org member & read 200+ pages of spec)
  - Read our 12 golden rules of competent graph systems design (just 1 page of reading)
  - See how DuckDB extensibility can be leveraged for a modular implementation of SQL/PGQ
    (..and we also present some benchmark results..)

- DuckPGQ availability? Not yet.. WIP & ETA in 2023

- Many avenues for future data systems research:
  - Factorized query execution, Vectorized WCOJs & their query optimization
  - Path-finding and query optimization, better path-finding parallelism