Introduction to Graph Databases

Activity – Implementing Graphs using Relational Databases

Google released, in 2002, a subset of the structure of the WWW. In this dataset, web pages are represented by graph nodes such that when a web page A contains a hyperlink to web page B, a directed edge is created from node A to node B.

In this activity, we will focus on the performance of different queries. Therefore, we will use three PostgreSQL tables, which are subsets of different sizes of the web structure released by Google:

- **webgraph1** table: 605 nodes (web pages) and 1521 edges (hyperlinks)
- **webgraph2** table: 1622 nodes (web pages) and 6288 edges (hyperlinks)
- **webgraph3** table: 4122 nodes (web pages) and 14356 edges (hyperlinks)

If you are using your own local PostgreSQL database, you need to create and populate it. Use the three files provided.

```
CREATE TABLE  webgraph1 (fromnode int,  tonode int);

COPY webgraph1   FROM   'X:..\webgraph1.txt'


CREATE TABLE  webgraph2 (fromnode int,  tonode int);

COPY webgraph2   FROM   'X:..\webgraph2.txt'


CREATE TABLE  webgraph3 (fromnode int,  tonode int);

COPY webgraph3   FROM   'X:..\webgraph3.txt'
```

Below, we show the list of different uses cases we want to analyze:

Use Case A: For each pair of connected nodes, find the **1-hop paths**. Include four columns in the resultset: **fromNode, toNode, length, path**, which correspond to the source node, target node, length of the path, and visited nodes, respectively. Exclude repeated nodes in the path. That is, if A -> A, do not consider that A, A, 1, A-A is a valid 1-path from A to A.

Use Case B: For each pair of connected nodes, find the **2-hop paths**. Include four columns in the resultset: **fromNode, toNode, length, path**, which correspond to the source node, target node, length of the path and the visited nodes, respectively. Exclude repeated nodes in the path. That is, if A -> B and B->A, do not consider that A, A, 2, A-B-A is a valid 2- path from A to A.

Use Case C: For each pair of connected nodes, find the **3-hop paths**. Include four columns in the resultset: **fromNode, toNode, length, path**, which correspond to the source node, target node, length of the path and the visited nodes, respectively. Exclude repeated nodes in the path. That is, if A -> B, A->C and B->A, do not consider that A, C, 3, A-B-A-C is a valid 3-path from A to C.

Use Case D: for each pair of connected nodes, find the **N-hop paths (the value of N is not known in advance)**. Include four columns in the result set: **fromNode, toNode, length, path**, which correspond to the source node, target node, length of the path and the visited nodes, respectively. Exclude repeated nodes in the path like in case "C".

Exercise 1

1.1) Show the SQL queries that solve each use case. Use an alias for the table, such that it can be easily rewritten for different table names.

Use Case	SQL (use an alias for the table in the from clause – Replace webgraph1 as needed)
A (1-hop)	select wg.fromnode, wg.tonode, 1 long, wg.fromnode ||'-'|| wg.tonode path from webgraph1 wg where wg.fromnode <> wg.tonode
B (2-hop)	select w1.fromnode, w2.tonode, 2 long, w1.fromnode ||'-'|| w2.fromnode || '-' || w2.tonode as path from webgraph3 w1, webgraph3 w2 where w1.tonode = w2.fromnode and w1.fromnode <> w1.tonode and w2.fromnode <> w2.tonode and w1.fromnode <> w2.tonode

C (3-hop)	select w1.fromnode as initial, w1.tonode ||'-'|| w2.tonode as path, w3.tonode as final, 3 length from webgraph3 w1, webgraph3 w2, webgraph3 w3 where w1.tonode = w2.fromnode and w2.tonode = w3.fromnode and w1.fromnode <> w1.tonode and w2.fromnode <> w2.tonode and w3.fromnode <> w3.tonode and w1.fromnode <> w2.tonode and w1.fromnode <> w3.tonode and w2.fromnode <> w3.tonode
D (N-hop)	with recursive auxi(fromnode, tonode, long, path) as (select wg.fromnode, wg.tonode, 1, '-'|| wg.fromnode ||'-'|| wg.tonode from webgraph1 wg where wg.fromnode <> wg.tonode union select auxi.fromnode, wg.tonode, 1 + long , path ||'-'|| wg.tonode from auxi, webgraph1 wg where auxi.tonode = wg.fromnode AND wg.fromnode <> wg.tonode AND position('-'|| wg.tonode || '-' in auxi.path) = 0) select * from auxi

1.2) Run each query in Part 1.1., against tables of different sizes. For each run, record the execution time and the size of the result set (number of tuples), and complete the following comparative tables.

Use Case A (1-hop)		
Table	Execution Time (msec)	Resultset Size (#tuples)
1521 tuples		1521
6288 tuples		6288
14356 tuples		14356

Use Case **B** (2-hop)		
Table	Execution Time (msec)	Resultset Size (#tuples)
1521 tuples		10164
6288 tuples		60954

14356 tuples		138318

Use Case C (3-hop)		
Table	Execution Time (msec)	Resultset Size (#tuples)
1521 tuples		90393
6288 tuples		649111
14356 tuples		1647065

Use Case D (N-hop)		
Table	Execution Time (msec)	Resultset Size (#tuples)
1521 tuples		?
6288 tuples		?
14356 tuples		?

Exercise 2

2.1) Probably, some queries above will run indefinitely. Analyze the strategy used by PostgreSQL for executing each query. More precisely, find out the query plan chosen for the largest table (webgraph3), for each one of the use cases. Complete the following table.

Answer: Run EXPLAIN SELECT ….

Use case	Query Plan for webgraph3
A (1-hop)	Seq Scan on webgraph3 wg (cost=0.00..457.71 rows=14284 width=44) Filter: (fromnode <> tonode)
B (2-hop)	Hash Join (cost=211.31..2246.32 rows=33130 width=44) Hash Cond: (w1.tonode = w2.fromnode) Join Filter: (w1.fromnode <> w2.tonode) -> Seq Scan on webgraph3 w1 (cost=0.00..133.85 rows=6197 width=8) Filter: (fromnode <> tonode) -> Hash (cost=133.85..133.85 rows=6197 width=8) -> Seq Scan on webgraph3 w2 (cost=0.00..133.85 rows=6197 width=8) Filter: (fromnode <> tonode)

C (3-hop)	…
D (N-hop)	…

2.2) Briefly sketch the idea behind the query plans proposed by PostgreSQL .

Needs to JOIN tables.
For this, always a TABLE SCAN is performed. Then, sometimes a Hash is performed. If there are multiple joins, the stategy normally is to run MERGE operations pairwise. For this, both tables must be sorted, thus a SORT is run. As the comparison between join attributes is done, a FILTER is applied for the other conditions.

Exercise 3

In Exercise 1, probably **N-hop** queries over table **webgraph3** ran indefinitely. However, note that a 3-hops query could also be solved by an SQL recursive query limited to N=3.

3.1) Rewrite the **recursive** SQL query **over the webgraph3 table**, limiting it to retrieve **only 3-Hops.** Verify the result, checking that you obtained the same results as with the 3-hop non-recursive version, that is, your SQL recursive query limited to N=3 is equivalent to a non-recursive triple join query.

```
with recursive auxi(fromnode, tonode, long, path) as
(
   select wg.fromnode, wg.tonode, 1,  wg.fromnode  ||'-'||  wg.tonode
   from webgraph3 wg
   where wg.fromnode <> wg.tonode
   union
   select auxi.fromnode, wg.tonode, 1 + long  , path  ||'-'||  wg.tonode
          from auxi, webgraph3 wg
          where auxi.tonode = wg.fromnode
          AND wg.fromnode <> wg.tonode
          AND  position('-'||  wg.tonode || '-'  in auxi.path ) = 0 AND long < 3
)
select * from auxi where long = 3
```

3.2) Now, we want to study if the execution time could be improved using indexes. We will take into consideration both query variations, i.e. 3-Hops (triple join), and SQL recursive limited to obtain 3-Hops.

3.2.1) Which index could be useful for avoid the TableScan+Sort? Which index would be useful for the Merge Operator? Write the SQL syntax for creating the index/es proposed.

Answer:

create index f1 on webgraph3(fromnode)

create index f2 on webgraph3(tonode)

3.2.2) Create the index/es proposed. Run both queries. Compare the result against the 3-hop performance in exercise 1. Complete the following table. Did you obtain any improvement?

Use Case C (3-hop)		
Table with index/es	Execution time (sec)	Resultset size (#tuples)
14356 tuples		

Use Case D (N-hop) - SQL recursive limited by N=3		
Table with index/es	Execution time (sec)	Resultset size (#tuples)
14356 tuples		

3.3.3) Analyze the query plan for both queries (with indexes). Complete the following table:

Use case	Query plan with index/es for webgraph3
C (3-hop)	
D (N-hop) limited by N=3	

3.3.4) Did PostgreSQL use the same strategy in both queries? Explain the reason in detail.

Answer
Typically, in the 3-hop query, both indexes should be used. It doesn't require a table scan and sort. The merge is still performed.
In the N-Hop query, the index can only be used in the first iteration. Since the recursive table is a temporary one, there is no index on it. It must be sorted each time a MERGE is performed. Thus, the index doesn't help here.

Exercise 4

When looking for N-paths, where N is large (>3) or unknown, we need to use Recursive SQL. Execute the recursive SQL query and find the **longest N value** that allows us to obtain a result set in **less than 5 minutes**, for the **webgraph3** table.

Answer: