Introduction to Graph Data Management

Claudio Gutierrez

Center for Semantic Web Research (CIWS)
Department of Computer Science
Universidad de Chile

EDBT Summer School – Palamos 2015
Joint Work With

Renzo Angles
Universidad de Talca, Chile
Agenda for today: querying

1. Reminder / comment on first lecture
2. Graph query language concepts
3. Querying graphs
4. Graph database and systems
Golden Age of Graph Databases

Jan Hidders

Alexandra Poulouvassilis

Golden age
Reminder I: Property Graph data model
Graph Theory (R. Diestel)
1. Introduction
2. Matching
3. Connectivity
4. Planar Graphs
5. Colouring
6. Flows
7. Substructures in Dense Graph
8. Ramsey Theory for Graphs
9. Hamilton Cycles
10. Random Graphs
11. Minor, Trees, Well Quasi Orders

Networks: An Introduction (M. Newman)
1. Introduction
2. Technological Netowrks
3. Social Networks
4. Networks of information
5. Biological Networks
6. Mathematics of Networks
7. Measures and Methods
8. The large-scale structure of networks
9. Basic concepts of algorithms
10. Fundamental network algorithms
11. Matrix algorithms and graph partitioning
12. Random Graphs
13. Random Graphs with general degree distribution
14. Models of network formation
15. Other network models
16. Percolation and network resilience
17. Epidemics on networks
18. Dynamical system on networks
19. Network search
Q1 (Property Graph data model)
Name one positive feature and one negative feature of the Property Graph data model

Q2 (Graph theory – Data management)
Name one result (theorem, area, topic, algorithm, technique, etc.) from Graph Theory that you consider could be useful for improving Graph Data management.
Graph Query Language Notions
Database Models: Codd’s definition

Data structures

Integrity constraints

Query Language
Data manipulation is expressed by graph transformations, or by operations whose main primitives are on graph features like paths, neighborhoods, subgraphs, graph patterns, connectivity, and graph statistics.
A supermarket list of types of queries

A. “Basic” Graph Queries
1. Pattern matching
2. Adjacency / neighborhood
3. Reachability / connectivity
   1. Regular (and regular++)
   2. CRPQ
   3. etc.
4. Summarization
5. …
A supermarket list of types queries (cont.)

B. Analytical Queries
1. Centrality measures
2. Diameter and other global properties
3. Various statistics
4. Graph properties and parameters
5. …
Something is getting wrong…

Seems like we are in Linnean times: lots of arbitrary animals collected and discovered, but no way of making sense of this diversity

Either:

- we are not understanding graphs

or

graphs are not understandable by XXI’s century humans

or

we do not know what we are looking for …

but one thing is clear: a scientific description cannot be an arbitrary list of properties
Some desirable features of a query language

1. Genericity (independence of coding of data)
2. Good expressive power
3. Low complexity of evaluation
4. Simple syntax and semantics
5. Compositionality
6. Few and simple constructors
7. Hopefully not operational semantics
8. User friendly / low barrier of entrance
9. Standard…
Graph Query Language: I/O types
Graph Query Language: basic modules

- Extract
- Define data sources
- Transform
- Construct
Graph query languages: their basic modules

<table>
<thead>
<tr>
<th>Language</th>
<th>Define Source</th>
<th>Extract</th>
<th>Transform</th>
<th>Construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>FROM</td>
<td></td>
<td>WHERE</td>
<td>SELECT</td>
</tr>
<tr>
<td>SPARQL</td>
<td>FROM, Service</td>
<td>pattern matching</td>
<td>operators</td>
<td>Select, ASK, Construct,</td>
</tr>
<tr>
<td>Datalog</td>
<td></td>
<td>match facts</td>
<td>rules</td>
<td>head</td>
</tr>
<tr>
<td>XQuery</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSLT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Exercise:** Fill in the blanks; add your favorite language
SPARQL Query

Query Form
- CONSTRUCT
- DESCRIBE
- SELECT
- ASK

Dataset Clause
- FROM
- FROM NAMED

WHERE Clause (Graph Pattern)
- Triple pattern
- OPTIONAL
- AND
- UNION

TRUE - FALSE
MATCH (p:Person)-[:Knows]->(friend)
WHERE p.age = 20
WITH p, count(friend) as friends
WHERE friends > 0
RETURN p.name, friends
## Cypher Query Language: outputs

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A node</td>
<td>MATCH (p:Person {name:&quot;Tom&quot;}) RETURN p</td>
</tr>
<tr>
<td>A value</td>
<td>MATCH (p:Person {name:&quot;Tom&quot;}) RETURN p.age</td>
</tr>
<tr>
<td>A list of values</td>
<td>MATCH (p:Person) RETURN p.name LIMIT 5</td>
</tr>
<tr>
<td>An array</td>
<td>MATCH p=shortestPath((a)-[*]-&gt;(b)) WHERE a.name=&quot;Axel&quot; AND b.name=&quot;Tom&quot; RETURN p</td>
</tr>
<tr>
<td>A list of arrays</td>
<td>MATCH p=((a)-[*]-&gt;(b)) WHERE a.name=&quot;Axel&quot; AND b.name=&quot;Frank&quot; RETURN p</td>
</tr>
</tbody>
</table>
Each social application is a consumer/producer of social networks, producing and/or collecting network data, and consuming data produced by other applications. [SNQL, SanMartin,_,Wood]
An aside: a different problem or "the" problem?

The web is one more artifact or is "the" answer to scalability?
The “use case” that triggered the Web design
1. Data sources/services are reliable
2. Consumer behaviour can be anticipated
3. Publishers are infallible and play no role
4. You can know what’s out there
5. Universal cost models can be maintained
6. Query execution is always deterministic
7. Standards = interoperability
8. One system can ACE them all
   (ACE: alignment, coverage, efficiency)
Graph Databases and Systems
Native Data Store

APIs

Applications

Services

Data Structure: Graphs

Query languages

Native Data Store

Files

RDBMS

Oracle

DB2

MySQL

MSQL

Postgres
## Classification (most influential models)

<table>
<thead>
<tr>
<th>Database model</th>
<th>Abstraction level</th>
<th>Data structure</th>
<th>Information focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Physical</td>
<td>Pointers, records</td>
<td>Records</td>
</tr>
<tr>
<td>Relational</td>
<td>Logical</td>
<td>Relations</td>
<td>Data, attributes</td>
</tr>
<tr>
<td>Semantic</td>
<td>User</td>
<td>Graph</td>
<td>Schema, relations</td>
</tr>
<tr>
<td>OO</td>
<td>Physical/logical</td>
<td>Objects</td>
<td>Objects, methods</td>
</tr>
<tr>
<td>Semi-structured</td>
<td>Logical</td>
<td>Tree</td>
<td>Data, components</td>
</tr>
<tr>
<td>Graph</td>
<td>Logical/user</td>
<td>Graph</td>
<td>Data, relations</td>
</tr>
</tbody>
</table>
Classification issues (taken from P. Boncz’s lecture)

(Interactive, BI, Graph analytics)

Graph Databases
Graph programming frameworks
RDF databases
Relational databases
NoSQL Key-value
NoSQL MapReduce
Batch processing

…
Classification issues (taken from B. Shao’s lecture)

Offline processing (offline analytics)

Online processing (online querying)

Optimized for response time or throughput

Transactional

…
1. Address need of managing graph data
2. Architecture/goals inspired by classical DBMS
3. Persistent storage of graph data
4. Transactionality
5. Closed world
6. Efficiency (over scalability)
7. (Near future:) Portability (of data)
8. (Near future:) Declarative query languages
## Graph Databases

<table>
<thead>
<tr>
<th>Graph database</th>
<th>Data model</th>
<th>Storage method</th>
<th>Query facilities</th>
<th>Computing model</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>ArangoDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Bitsy</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Cayley</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>FlockDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>GraphBase</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Graphd</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Horton</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>IBM System G</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>imGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfoGrid</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>OrientDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sparksee/DEX</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Titan</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Trinity</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>TurboGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
Graph Processing Frameworks

(Offerline Graph Analytical Systems)

1. Batch processing
2. Analysis of large graphs
3. Facilities for graph analytical algorithms
4. Distributed environment
5. Multiple machines
6. API or programming as user access
Graph Processing Frameworks / Offline graph analytical syst.

- Pregel
- Apache Giraph
- GraphLab
- Catch the Wind
- GPS
- Mizan
- Power Graph
- GraphX
- TurboGraph
- GraphChi
- …
Some conclusions / opinions

• Graph data management has a bright future: we are living very interesting times
• One size does not fit all: Need different GDB for small, medium and web scale: marry one, get aware of your choice, and learn to love it… try not to flirt with others.
• Not evident that there will be one standard graph query language: too diverse use cases.
• Need to better understand graphs
• What matters in a graph is its topology [if you do not need it, stay in the relational world]
• Need better interoperation between relational (tables) and graph data.