An Algebra and Equivalences to Transform Graph Patterns in Neo4j

Data model: Property graphs and graph relations

- Labeled edges and nodes
- Every edge and node carries a set of properties
- Plain old relational algebra can be used on the graph relations

The algebra: *getNodes* and *expand*

- *getNodes*: Selects nodes from (sub)graph, based on attribute x
- Returns a graph relation
- *expand*: Concatenates neighbours of a selected node to a relation
- *expandIn* for ingoing edges, *expandOut* for outgoing ones

An Algebra and Equivalences to Transform Graph Patterns in Neo4j

Optimisation - the equivalence rules

- Finding outgoing edges from x to y is the same as finding ingoing edges for y from x
- $expandOut(x, y) \circ getNodes(x) \equiv expandIn(y, x) \circ getNodes(y)$
- Generalised to three nodes:
- $expandOut(y, z) \circ expandOut(x, y) \circ getNodes(x) \equiv$ $expandOut(y, x) \circ expandOut(z, y) \circ getNodes(z)$
- Finding ingoing edges from two nodes x, z to y:
- $expandIn(z, y) \circ expandIn(x, y) \circ getNodes(y) \equiv$ $expandIn(x, y) \circ expandOut(z, y) \circ getNodes(z)$
- Expansions can be interchanged if they are independent (for $A, B \in \{in, out\}$):
- $expandA(a, b) \circ expandB(c, d) \equiv expandB(c, d) \circ expandA(a, b)$

An Algebra and Equivalences to Transform Graph Patterns in Neo4j

- Joining traversals starting from multiple nodes (where E is a graph relation):
- $expandIn(y, z) \circ expandOut(x, y)(E) \equiv expandOut(x, y)(E) \bowtie expandOut(z, y) \circ getNodes(z)$
- Selection can be "pushed" inwards, under an expansion operator - provided that no variable of selection is introduced by expansion:
- $\sigma_F \circ expand(x, y)(E) \equiv expand(x, y) \circ \sigma_F(E)$

An Algebra and Equivalences to Transform Graph Patterns in Neo4j

- Optimisation of Cypher queries, empirically - no cost model yet
- No support for: variable length patterns, node aggregation
- Selection can be "pushed" inwards, under an expansion operator - provided that no variable of selection is introduced by expansion:
- $\sigma_F \circ expand(x, y)(E) \equiv expand(x, y) \circ \sigma_F(E)$

An Algebra and Equivalences to Transform Graph Patterns in Neo4j

- Joining traversals starting from multiple nodes (where E is a graph relation):
- $expandIn(y, z) \circ expandOut(x, y)(E) \equiv expandOut(x, y)(E) \bowtie expandOut(z, y) \circ getNodes(z)$
- Selection can be "pushed" inwards, under an expansion operator - provided that no variable of selection is introduced by expansion:
- $\sigma_F \circ expand(x, y)(E) \equiv expand(x, y) \circ \sigma_F(E)$

An Algebra and Equivalences to Transform Graph Patterns in Neo4j

- Optimisation of Cypher queries, empirically - no cost model yet
- No support for: variable length patterns, node aggregation
- Selection can be "pushed" inwards, under an expansion operator - provided that no variable of selection is introduced by expansion:
- $\sigma_F \circ expand(x, y)(E) \equiv expand(x, y) \circ \sigma_F(E)$

Other papers

- Foundations of RDF Databases - Arenas et al. : Full semantics and complete/sound inference calculus for RDF graphs
- RAL: an Algebra for Querying RDF - Frasincar et al. : Extraction operators (get data), loop operators (repeated application), construction operators (build RDF).
  Operators have the form: $o[f](x_1, x_2, \ldots : \textit{expression})$ - Bind x_i to inputs, compute $f(x_i)$, compute $o(f(x_i))$, combine via set union partial applications for the result

Other papers

- Algebra of RDF Graphs for Querying Large-Scale Distributed Triple-Store - Savnik et al. : A “physical” (practical?) algebra on shared-nothing clusters.
- A relational algebra for SPARQL - Cyganiak : From SPARQL to SQL - from RDF triples to RDF tuples (relations = sets of these tuples)