Incremental Graph Queries with openCypher

Gábor Szárnyas
GraphDevroom @ FOSDEM 2017

Budapest University of Technology and Economics
McGill University, Montréal
Incremental Graph Queries with openCypher
Incremental Graph Queries
with openCypher
Live railway model
Live railway model
Live railway model
Live railway model
Live railway model

Proximity detection
Live railway model

Proximity detection
Live railway model

Proximity detection
Live railway model

Proximity detection

Trailing the switch
Live railway model
Live railway model
Live railway model
Live railway model
Proximity detection

≤ 2 segments
Proximity detection

≤ 2 segments

NEXT: 1..2
Proximity detection

MATCH

(t1:Train)[[:ON]]->(seg1:Segment)
-[:NEXT*1..2]->(seg2:Segment)
<-[[:ON]]-(t2:Train)

RETURN t1, t2, seg1, seg2
Proximity detection

MATCH

(t1:Train)-[:ON]->(seg1:Segment)-[:NEXT*1..2]->(seg2:Segment)<-[::ON]-(t2:Train)

RETURN t1, t2, seg1, seg2
Trailing the switch
Trailing the switch
MATCH (t:Train)-[:ON]->(seg:Segment)
<-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment)<-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg)-[:STRAIGHT]->(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
Incremental queries
Incremental queries

- Register a set of **standing queries**
- **Continuously evaluate** queries on changes
Incremental queries

- Register a set of **standing queries**
- **Continuously evaluate** queries on changes

- The **Rete algorithm** (1974)
  - Originally for rule-based expert systems
  - Indexes the graph and caches interim query results
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position}} = '\text{diverging}' \} \]

ON ─────────────────────────────────────────── STRAIGHT
Trailing the switch

\[ \pi_{t.number, sw} \]

\[ \sigma_{sw.position = 'diverging'} \]

1

2

a

b

c

d

e

div

f

g

NEXT

STRAIGHT

DIVERGING

TOP
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'}{}} \]

\[ \bowtie \]

\[ \text{ON} \]

\[ \text{STRAIGHT} \]
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = 'diverging'} \]

ON

STRAIGHT

1

2

a \to b \to c \to d \to e \to f

g \to \text{div} \to \text{TOP}
Trailing the switch

$\pi_{t.number, sw}$

$\sigma_{sw\text{.position} = \text{'diverging'}}$

ON

1 \rightarrow a 
2 \rightarrow e 

STRAIGHT

$\bowtie$

1 \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e 
2 \rightarrow e \rightarrow f 

NO

NEXT

NO

STRAIGHT

TOP

DIVERGING
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = '\text{diverging'} } \]

STRAIGHT ON a 1 ON a 2 ON e

ON 1 ON a 2 ON e

STRAIGHT
Trailing the switch

$\pi_{t.number, \text{sw}}$

$\sigma_{\text{sw.position} = \text{'diverging'}}$

$\bowtie$

STRAIGHT

ON 1 ON a 2 ON e

STRAIGHT

ON

ON

ON

ON

ON

STRAIGHT

div
g

f

div

NEXT

NEXT

NEXT

NEXT

NEXT

STRAIGHT

TOP

DIVERGING
Trailing the switch

π_{t.number, sw}

σ_{sw.position = 'diverging'}
Trailing the switch

\[ \pi_{t\.number, sw} \]

\[ \sigma_{\text{sw\.position} = \text{'diverging'}} \]

N

STRAIGHT

div

ON

1 2 ON e

STRAIGHT e div

ON a 2 ON e

NEXT a b c d e

STRAIGHT div f

NEXT g
Trailing the switch

\[ \pi_{\text{t.number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = \text{'diverging'}} \]

Diagram:

- Node 1 (ON) to Node a (NEXT)
- Node a (NEXT) to Node b (NEXT)
- Node b (NEXT) to Node c (NEXT)
- Node c (NEXT) to Node d (NEXT)
- Node d (NEXT) to Node e (NEXT)
- Node e to Node div (STRAIGHT)
- Node div to Node f (TOP)
- Node e (DIVERGING) to Node g (DIVERGING)
Trailing the switch

\[ \pi_{t.\text{number, } sw} \]

\[ \sigma_{\text{sw.position} = '\text{diverging'} \}

\text{STRAIGHT ON}
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'} } \]

1. ON
2. STRAIGHT
Trailing the switch

\[ \pi_{t\text{.number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = '\text{diverging'} } \]

1. ON a
   - ON 2
   - ON e

2. STRAIGHT e
   - STRAIGHT div

ON 1

NEXT a → b → c → d → e

ON 2

NEXT e → div → TOP f

NEXT g

STRAIGHT e

DIVerging
Trailing the switch

\[ \pi_{t.number, \text{sw}} \]

\[ \sigma_{\text{sw.position} = '\text{diverging'} } \]

\[ \text{ON} \quad 1 \quad \text{ON} \quad a \quad 2 \quad \text{ON} \quad e \]

\[ \text{STRAIGHT} \quad e \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]

\[ \text{div} \quad \text{STRAIGHT} \quad \text{div} \]
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = \text{'diverging'}} \]

1. ON → a → 2 → ON → e
2. STRAIGHT → e → STRAIGHT → div

1. NO → a → NEXT → b → NEXT → c → NEXT → d → NEXT → e
2. NO → e → STRAIGHT → div → TOP → f

\[ g \]
Trailing the switch

\[ \pi_{t.\text{number, sw}} \]

\[ \sigma_{\text{sw.position = 'diverging'} } \]

ON

STRAIGHT

NEXT

DIVerging
Trailing the switch

\[ \pi_{t\text{.number}, \text{sw}} \]

\[ \sigma_{\text{sw.position} = \text{'diverging'}} \]

NEXT

ON

STRAIGHT

ON

STRAIGHT

ON

STRAIGHT

STRAIGHT

TOP

DIVERGING
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position}} = 'diverging'} \]

ON

STRAIGHT

NEXT

DIVERGING

TO

P

div

div

div

div

div

div

div

div

div
Trailing the switch

\[ \pi_{t\text{.number, } sw} \]

\[ \sigma_{sw\text{.position} = \text{'diverging'}} \]

\[ \bowtie \]

ON

STRAIGHT

\[ \text{div} \]

\[ \text{ON} \]

\[ \text{STRAIGHT} \]

\[ \text{div} \]

\[ \text{ON} \]

\[ \text{STRAIGHT} \]

\[ \text{div} \]

\[ \text{ON} \]

\[ \text{STRAIGHT} \]

\[ \text{div} \]

\[ \text{ON} \]

\[ \text{STRAIGHT} \]

\[ \text{div} \]

\[ \text{ON} \]

\[ \text{STRAIGHT} \]

\[ \text{div} \]
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position}} = \text{'diverging'}} \]

\[ g \]

\[ e \]

\[ d \]

\[ c \]

\[ b \]

\[ a \]
Trailing the switch

\[ \pi_{t.number, sw} \]

\[ \sigma_{sw.position = 'diverging'} \]

ON

1 → a → 2 → d

STRAIGHT

e ← div

\[ a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow div \rightarrow f \]

\[ g \leftarrow \text{DIVERGING} \]
Trailing the switch

π_{t.number, sw}

σ_{sw.position = 'diverging'}
Batch vs. incremental queries

- **Batch queries** (pull / request-driven):
  1. Client selects a query
  2. Results are calculated

- Query results obtained on demand
Batch vs. incremental queries

- **Batch queries**
  (pull / request-driven):
  1. Client selects a query
  2. Results are calculated

- Query results obtained on demand

- **Incremental queries**
  (push / event-driven):
  1. Client registers queries
  2. Graph is changed
  3. Results are maintained
  4. Goto 2

- Query results are always available
Incremental query engines

- **CLIPS**
  - C structures
  - NASA
- **Drools**
  - POJO
  - Red Hat
- **VIATRA**
  - EMF
  - BME / IncQuery Labs.
Incremental query engines

- **CLIPS**
  - Structures: C structures
  - Provider: NASA

- **Drools**
  - Data Model: POJO
  - Provider: Red Hat

- **VIATRA**
  - Data Model: EMF
  - Provider: BME / IncQuery Labs.

- **INSTANS**
  - Data Model: RDF
  - Provider: Aalto University

- **i3QL**
  - Data Model: POJO
  - Provider: TU Darmstadt

- **IncQuery-D**
  - Data Model: RDF
  - Provider: BME
### Incremental query engines

<table>
<thead>
<tr>
<th>Tool</th>
<th>Data Model</th>
<th>Implementor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIPS</td>
<td>C structures</td>
<td>NASA</td>
</tr>
<tr>
<td>Drools</td>
<td>POJO</td>
<td>Red Hat</td>
</tr>
<tr>
<td>VIATRA</td>
<td>EMF</td>
<td>BME / IncQuery Labs.</td>
</tr>
<tr>
<td>INSTANS</td>
<td>RDF</td>
<td>Aalto University</td>
</tr>
<tr>
<td>i3QL</td>
<td>POJO</td>
<td>TU Darmstadt</td>
</tr>
<tr>
<td>IncQuery-D</td>
<td>RDF</td>
<td>BME</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>No implementations for property graphs yet</strong></td>
</tr>
</tbody>
</table>
Incremental Graph Queries with openCypher
Incremental Graph Queries with openCypher
The openCypher project aims to deliver a full and open specification of the industry’s most widely adopted graph database query language: Cypher.

- Grammar specification
- Technology Compatibility Kit (TCK)
- Reference implementation
openCypher constructs

- Standard constructs
  - pattern matching
  - filtering
  - lists, maps
  - data manipulation
  - variable length paths
openCypher constructs

- **Standard constructs**
  - pattern matching
  - filtering
  - lists, maps
  - data manipulation
  - variable length paths

- **Legacy constructs**
  - indexing, constraints
  - regular expressions
  - some list functions, including reduce
  - most predicate functions
  - shortest path functions
  - CASE expressions
  - id()
openCypher constructs

- Standard constructs
  - pattern matching
  - filtering
  - lists, maps
  - data manipulation
  - variable length paths

- Legacy constructs
  - indexing, constraints
  - regular expressions
  - some list functions, including reduce
  - most predicate functions
  - shortest path functions
  - CASE expressions
  - id()

Difficult to handle incrementally
### Mapping openCypher to relational algebra

**Combining and filtering pattern matches**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH «p»</td>
<td>( \exists \text{edges of } p ) (p)</td>
</tr>
<tr>
<td>MATCH «p1», «p2»</td>
<td>( \exists \text{edges of } p1 \text{ and } p2 ) (p1 ( \times ) p2)</td>
</tr>
<tr>
<td>MATCH «p1»</td>
<td>( \exists \text{edges of } p1 ) (p1) ( \times ) ( \exists \text{edges of } p2 ) (p2)</td>
</tr>
<tr>
<td>MATCH «p2»</td>
<td>( \exists \text{edges of } p2 ) (p2)</td>
</tr>
<tr>
<td>MATCH «p1»</td>
<td>( \exists \text{edges of } p1 ) (p1) ( \times ) ( \exists \text{edges of } p2 ) (p2)</td>
</tr>
<tr>
<td>OPTIONAL MATCH «p2»</td>
<td>( \exists \text{edges of } p1 ) (p1) ( \times ) ( \exists \text{edges of } p2 ) (p2)</td>
</tr>
<tr>
<td>MATCH «p»</td>
<td>( \sigma_{\text{condition}}(r) ), where condition may specify patterns and arithmetic constraints on existing variables</td>
</tr>
<tr>
<td>WHERE «condition»</td>
<td></td>
</tr>
</tbody>
</table>

**Result and sub-result operations. Rules for RETURN also apply to WITH.**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN «variables»</td>
<td>( \pi_{\text{variables}}(r) )</td>
</tr>
<tr>
<td>RETURN «v1» AS «alias1» ...</td>
<td>( \pi_{v1 \rightarrow \text{alias1},...}(r) )</td>
</tr>
<tr>
<td>RETURN DISTINCT «variables»</td>
<td>( \delta(\pi_{\text{variables}}(r)) )</td>
</tr>
<tr>
<td>RETURN «variables», «aggregates»</td>
<td>( \gamma_{\text{variables,aggregates}}(r) )</td>
</tr>
</tbody>
</table>

**List operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER BY «v1» [ASC</td>
<td>DESC] ...</td>
</tr>
<tr>
<td>LIMIT «l»</td>
<td>( \lambda_{l}(r) )</td>
</tr>
</tbody>
</table>
## Mapping openCypher to relational algebra

### Combining and filtering pattern matches

<table>
<thead>
<tr>
<th>MATCH</th>
<th>$\exists$ edges of $p_1 \land p_2 \left( p_1 \bowtie p_2 \right)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH $\langle p \rangle$</td>
<td>$\exists$ edges of $p$</td>
</tr>
<tr>
<td>MATCH $\langle p_1 \rangle$, $\langle p_2 \rangle$</td>
<td>$\exists$ edges of $p_1$ and $p_2$</td>
</tr>
<tr>
<td>MATCH $\langle p_2 \rangle$</td>
<td>$\exists$ edges of $p_2$</td>
</tr>
<tr>
<td>MATCH $\langle p_1 \rangle$, OPTIONAL MATCH $\langle p_2 \rangle$</td>
<td>$\exists$ edges of $p_1$ and $p_2$</td>
</tr>
<tr>
<td>MATCH $\langle p \rangle$, WHERE $\langle$condition$\rangle$</td>
<td>$\sigma_{condition}(r)$, where condition may specify patterns and arithmetic constraints on existing variables</td>
</tr>
</tbody>
</table>

### Result and sub-result operations. Rules for RETURN also apply to WITH.

<table>
<thead>
<tr>
<th>RETURN $\langle$variables$\rangle$</th>
<th>$\pi_{variables}(r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN $\langle v_1 \rangle$ AS $\langle$alias1$\rangle$ ...</td>
<td>$\pi_{v_1 \rightarrow \text{alias1}, ...}(r)$</td>
</tr>
<tr>
<td>RETURN DISTINCT $\langle$variables$\rangle$</td>
<td>$\delta \left( \pi_{variables}(r) \right)$</td>
</tr>
<tr>
<td>RETURN $\langle$variables$\rangle$, $\langle$aggregates$\rangle$</td>
<td>$\gamma_{\text{variables,aggregates}}(r)$</td>
</tr>
</tbody>
</table>

### List operations

| ORDER BY $\langle v_1 \rangle$ [ASC | DESC] ... | $\tau_{\uparrow/\downarrow v_1}$ |
|--------------------------|------------------|
| LIMIT $\langle l \rangle$ | $\lambda_l(r)$ |
MATCH (tom:Person {name: "Tom Hanks"})-[[:ACTED_IN]->(m)<-[:ACTED_IN]-]-(coActors)
RETURN coActors.name
MATCH (tom:Person {name: "Tom Hanks"})-[[:ACTED_IN]->(m)<-[:ACTED_IN]-](coActors)
RETURN coActors.name
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)
RETURN coActors.name
Incremental openCypher challenges

- Lists
  - ['a', 1, 2, true]
  - ['a', [1, [2]], true]
  - UNWIND
  - collect()

- Bag semantics, ORDER BY, SKIP and LIMIT

- Legacy constructs
  - shortestPath(), allShortestPaths()
  - reduce()
Incremental openCypher challenges

- Lists
  - ['a', 1, 2, true]
  - ['a', [1, [2]], true]
  - UNWIND
  - collect()

- Bag semantics, ORDER BY, SKIP and LIMIT

- Legacy constructs
  - shortestPath(), allShortestPaths()
  - reduce()

These are not handled by traditional Rete implementations
An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
An incremental, in-memory graph query engine
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <-[:STRAIGHT]-(sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
MATCH (t:Train)-[:ON]->(seg:Segment) <->[:STRAIGHT]- (sw:Switch)
WHERE sw.position = 'diverging'
RETURN t.number, sw
Trailing the switch

\[ \pi_{\text{t.number, sw}} \]

\[ \sigma_{\text{sw.position = 'diverging'} \} \]

ON

STRAIGHT
Trailing the switch

\[ \pi_{t.\text{number}, \text{sw}} \]

\[ \sigma_{\text{sw.\text{position} = 'diverging'} \]
Trailing the switch

\[ \pi_{t\text{.number, } sw} \]

\[ \sigma_{sw\.position = \text{\textquoteleft}diverging\textquoteright} \]

Async messages

Actors
Trailing the switch

\[ \pi_{t.number, \, sw} \]

\[ \sigma_{sw.\, position = \text{'diverging'}} \]

Challenge: properties

Async messages

Actors
Live railway model (revisited)

Proximity detection

Trailing the switch
The Rete network is a directed acyclic graph

Node reuse

π_{t, seg} (t, seg)

σ_{sw.position = 'diverging'} (t, seg, sw)

\bowtie (sw, seg) \rightarrow \pi_{t, number} (t, number, sw)

\bowtie [1..2] (t, seg, t, seg)

π_{t, seg} (t, seg)

\bowtie (seg1, seg2) \rightarrow \pi_{t} (t, seg1) \rightarrow \pi_{t} (t, seg2)

\bowtie (seg1:Segment) \rightarrow (seg2:Segment)

(t:Train) \rightarrow (seg:Segment)

\bowtie (seg1:Segment) \rightarrow (seg2:Segment)

\bowtie (seg1, seg2) \rightarrow (t1, t2, seg1, seg2)

\bowtie (t1, t2, seg1, seg2)

\bowtie (t1, t2, seg1, seg2)
ingraph use cases

Standing queries on large & quickly changing graphs

- Model validation: The Train Benchmark
- Static analysis of JavaScript source code
- Fraud detection
- IT infrastructure monitoring
Model validation: the Train Benchmark

- Scalable graph generator
  - EMF
  - Property graph
  - RDF
  - SQL
- Realistic workload
  - 6 validation queries
  - 12 transformations
  - Implemented for 12+ tools
- Visualization and reporting
var foo = 1 / 0;
var foo = 1 / 0;
Static analysis of JavaScript

```javascript
var foo = 1 / 0;
```

MATCH

```
(binding:BindingIdentifier)<-[[:binding]]-(()-->
(be:BinaryExpression)-[[:right]]->
(right:LNExpression)
```

WHERE be.operator = 'Div'
AND right.value = 0.0
RETURN binding
var foo = 1 / 0;

MATCH
(binding:BindingIdentifier)<-[[:binding]()]-->
(be:BinaryExpression)<-[[:right]]->
(right:LNExpression)
WHERE be.operator = 'Div'
AND right.value = 0.0
RETURN binding
```javascript
var foo = 1 / 0;
```

MATCH

(binding:BindingIdentifier)<-[[:binding]]->
(be:BinaryExpression)-[[:right]]->
(right:LNExpression)

WHERE be.operator = 'Div'
AND right.value = 0.0

RETURN binding

- Dead code detection
- Type inferencing
- Test generation
Project roadmap
Project roadmap

- Until now
  - formalization for most standard openCypher constructs
    - [technical report]
  - research prototype for model validation queries
Project roadmap

- Until now
  - formalization for most standard openCypher constructs [technical report]
  - research prototype for model validation queries
- 2017 Q1
  - Use TCK tests for feature coverage & debugging
Project roadmap

- **Until now**
  - formalization for most standard openCypher constructs
    [technical report]
  - research prototype for model validation queries

- **2017 Q1**
  - Use TCK tests for feature coverage & debugging

- **2017 Q2**
  - prototype for core openCypher constructs
  - implement optimizer
  - publish benchmark results [conference paper]
Project roadmap

- Until now
  - formalization for most standard openCypher constructs [technical report]
  - research prototype for model validation queries

- 2017 Q1
  - Use TCK tests for feature coverage & debugging

- 2017 Q2
  - prototype for core openCypher constructs
  - implement optimizer
  - publish benchmark results [conference paper]
  - Spark Catalyst or Apache Calcite
Open-Source Projects

Incremental Graph Engine:
https://github.com/ftsrg/ingraph

Train Benchmark:
https://github.com/ftsrg/trainbenchmark

BME-MODES3:
https://github.com/ftsrg/bme-modes3

Available under EPL v1.0.