Incremental View Maintenance for openCypher Queries

Gábor Szárnyas, József Marton
MODEL-DRIVEN ENGINEERING

- Primarily for designing critical systems
- Models are first class citizens during development
  - SysML / requirements, statecharts, etc.
  - Validation and code generation techniques for correctness

Technology:

Eclipse Modeling Framework (EMF)

- Originally started at IBM as an implementation of the Object Management Group’s (OMG) Meta Object Facility (MOF).
- i.e. an object-oriented model
- i.e. a property graph-like structure with a metamodel
MODEL VALIDATION

- Implemented with model queries
- Models are typed, attributed graphs

Typical queries

- Get two components connected by a particular edge
  
  \[
  \text{MATCH } (r:R)\ldots(s:S) \text{ WHERE NOT } (r)-[\text{:E}]->(s)
  \]

- Check if two objects are reachable
  
  \[
  \text{MATCH } (r:R)\ldots(s:S) \text{ WHERE NOT } (r)-[\text{:E1}\mid\text{E2*}]->(s)
  \]

- Property checks
  
  \[
  \text{MATCH } (r:R)\rightarrow(s:S) \text{ WHERE } r.a = 'x' \text{ OR } (s:Y)
  \]
RAILWAY NETWORK MODEL

IS THIS MODEL VALID?
MATCH (route:Route)
  -[:FOLLOWS]->(swP:SwitchPosition)
  -[:TARGET]->(sw:Switch)
  -[:MONITORED_BY]->(sensor:Sensor)
WHERE NOT (route)-[:REQUIRES]->(sensor)
RETURN route, sensor, swP, sw

G. Szárnyas, B. Izsó, I. Ráth, D. Varró: 
The Train Benchmark: cross-technology performance evaluation of continuous model queries. 
Software and Systems Modeling, 2017
route 1

sensor A

segment

segment

switch

switchPosition «diverging»

route 2

sensor B

segment

switch

switchPosition «straight»

sensor C

segment
INCREMENTAL VIEW MAINTENANCE (IVM)

In many use cases...
- queries are static
- data changes slowly

-> views can be maintained incrementally

Graph applications
- model validation
- simulation
- recommendation systems
- fraud detection
INGRAPH: IVM ON PROPERTY GRAPHS

**Idea:** map to relational algebra and use standard IVM techniques

- **Challenging aspects**
  - Property graph data model
  - Cypher language

- **Formalise the language in relational algebra**

- **Use nested relational algebra -> closed on operations**

**Prototype tool:** ingraph (OCIM1, OCIM2, GraphConnect talks)

---

Gábor Szárnyas, József Marton, Dániel Varró: *Formalising openCypher Graph Queries in Relational Algebra.*
ADBIS 2017
INGRAPH / GRAPH TO NESTED RELATIONS

```
<table>
<thead>
<tr>
<th>id</th>
<th>src</th>
<th>trg</th>
<th>type</th>
<th>properties</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>KNOWS</td>
<td>since 2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>c</td>
<td>INTEREST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>f</td>
<td>CLASS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>f</td>
<td>e</td>
<td>SUBCLASS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td>d</td>
<td>SUBCLASS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>label</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>name</td>
<td>Alice</td>
</tr>
<tr>
<td>Person</td>
<td>age</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>speaks</td>
<td>[en]</td>
</tr>
<tr>
<td>Person</td>
<td>name</td>
<td>Bob</td>
</tr>
<tr>
<td></td>
<td>age</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>speaks</td>
<td>[en, de]</td>
</tr>
<tr>
<td>Tag</td>
<td>topic</td>
<td>Neofolk</td>
</tr>
<tr>
<td>Class</td>
<td>subject</td>
<td>Folk</td>
</tr>
<tr>
<td>Class</td>
<td>subject</td>
<td>Music</td>
</tr>
<tr>
<td>Class</td>
<td>subject</td>
<td>Art</td>
</tr>
</tbody>
</table>
```
MATCH (p:Person)
OPTIONAL MATCH (p)-[i:INTEREST]->(t:Tag)
RETURN p.name, t

\[ \pi_{p.name, t} \bigcirc(p:Person) \exists_i^{(t:Tag)} [i:INTEREST] \]
INGRAPH

- ingraph uses a *procedural* IVM approach: the Rete algorithm.
  - Build caches for each operator
  - Maintain caches upon changes
  - Supports 15+ out of 25 LDBC BI queries
  - Details to be published in a conference paper
  - Extensible, but very heavy on memory

- The rest of the talk focuses on the *algebraic* approach.

---


arXiv preprint will be available on the 1\textsuperscript{st} week of June
Delta Queries for openCypher
DELTA QUERIES AT A GLANCE

evaluate query $Q$

for each $\Delta G$

evaluate $\Delta Q$

$\Delta G_1$, $\Delta G_2$, ...

changes $\Delta G_1$, $\Delta G_2$, ...

$\Rightarrow Q(G + \Delta G_1 + \Delta G_2 + \cdots)$

$Q$ and $\Delta Q$ are calculated by the same engine.
IMPLEMENTATION: TRIGGERS IN NEO4J

- Event-driven programming in databases
- Neo4j: TransactionEventHandler interface
  - afterCommit(TransactionData data, T state)
  - beforeCommit(TransactionData data)
  - TransactionData contains $\Delta G$: createdNodes, deletedNodes, ...
- Only the updated state of the graph is accessible.
- GraphAware framework: ImprovedTransactionData API
  - Get properties and labels/types of deleted elements

Michal Bachman: *Neo4j Improved Transaction Event API*. 2014

Max de Marzi: *Triggers in Neo4j*. 2015
DERIVING DELTA QUERIES

Idea: given query $Q$, derive delta queries $\Delta Q$ and $\nabla Q$, which define positive and negative changes, respectively.

But: most IVM techniques are defined for relational algebra.

Notation:
- $R$ relation
- $\Delta R$ positive changes
- $\nabla R$ negative changes
- $R^m$: maintained relation of $R \Rightarrow R^m = R - \nabla R + \Delta R$
- “−” denotes set minus (\), “+” denotes set union (∪)
RELATIONAL ALGEBRA FOR CYPHER

- Query plans in Neo4j $\cong$ relational algebra + Expand/VarExpand.
- Expand is essentially a natural join.

- Natural join $r \bowtie s$
- Semijoin $r \bowtie s = \pi_R(r \bowtie s)$
- Antijoin $r \bar{\bowtie} s = r \setminus (r \bowtie s)$
- Left outer join $r \bowtie s \cong (r \bowtie s) \cup (r \bar{\bowtie} s) \ // \text{plus nulls}$

Andrés Taylor: 

*Neo4j Cypher implementation.*
First openCypher Implementers Meeting, 2017
RELATIONAL ALGEBRA FOR CYPHER

Natural join: $r \bowtie s$
MATCH (v1)-[:r]->(v2)-[:s]->(v3)
RETURN *

Semijoin: $r \bowtie s$
MATCH (v1)-[:r]->(v2)
WHERE (v2)-[:s]->()

Antijoin: $r \ns\bowtie s$
MATCH (v1)-[:r]->(v2)
WHERE NOT (v2)-[:s]->()

Left outer join: $r \bowtie s$
MATCH (v1)-[:r]->(v2)
OPTIONAL MATCH (v2)-[:s]->(v3)
DERIVING DELTA QUERIES

X. Qian, G. Wiederhold: 
*Incremental Recomputation of Active Relational Expressions.*
TKDE 1991

T. Griffin, L. Libkin, H. Trickey: *An Improved Algorithm for the Incremental Recomputation of Active Relational Expressions.*
TKDE 1997

T. Griffin, L. Libkin: 
*Incremental Maintenance of Views with Duplicates.*
SIGMOD 1995

T. Griffin, B. Kumar: 
*Algebraic Change Propagation for Semijoin and Outerjoin Queries.*
SIGMOD Record 1998
DELTA QUERIES

- The seminal paper
- $\Delta/\forall$ delta queries for joins, selections, projections, etc.
- Bag semantics

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$\forall(t, Q)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$\forall R$, if $R \leftarrow (R \setminus \forall R) \cup \Delta R$ is in $t$, and $\phi$ otherwise</td>
</tr>
<tr>
<td>$\sigma_p(S)$</td>
<td>$\sigma_p(\forall(t, S))$</td>
</tr>
<tr>
<td>$\Pi_A(S)$</td>
<td>$\Pi_A(\forall(t, S)) \sqcup \Pi_A(\Delta(t, S))$</td>
</tr>
<tr>
<td>$S \uplus T$</td>
<td>$((\forall(t, S) \setminus \Delta(t, T)) \uplus (\forall(t, T) \setminus \Delta(t, S)))$</td>
</tr>
<tr>
<td>$S \ominus T$</td>
<td>$((\forall(t, S) \setminus \forall(t, T)) \uplus (\Delta(t, T) \setminus \Delta(t, S))) \min Q$</td>
</tr>
<tr>
<td>$S \min T$</td>
<td>$(\forall S \setminus (S \ominus T)) \max (\forall T \setminus (T \ominus S))$</td>
</tr>
<tr>
<td>$S \max T$</td>
<td>$((\forall(t, S) \uplus (\forall(t, T) \min (T \setminus \text{add}(t, S)))) \min (\forall(t, T) \uplus (\forall(t, S) \min (S \setminus \text{add}(t, T))))$</td>
</tr>
<tr>
<td>$\epsilon(S)$</td>
<td>$\epsilon(\forall(t, S)) \setminus \text{del}(t, S)$</td>
</tr>
<tr>
<td>$S \times T$</td>
<td>$((\forall(t, S) \times \forall(t, T)) \uplus ((\text{del}(t, S) \times \forall(t, T)) \setminus (\Delta(t, S) \times \text{del}(t, T))) \uplus$ $((\forall(t, S) \times \text{del}(t, T)) \setminus (\text{del}(t, S) \times \Delta(t, T))))$</td>
</tr>
</tbody>
</table>

DELTA QUERIES

- Semijoins, antijoins, outer joins
- Set semantics
- Later publications, e.g. Zhou-Larson’s ICDE’07 paper improved these

<table>
<thead>
<tr>
<th>Q</th>
<th>(\nabla Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S \bowtie T)</td>
<td>((S \bowtie (\nabla T \bowtie T^m)) \cup (\nabla S \bowtie (T - \nabla T)) \cup (\nabla S \bowtie (\Delta T \bowtie T)))</td>
</tr>
<tr>
<td>(S \sqcap T)</td>
<td>(((S - \nabla S) \bowtie (\Delta T \bowtie T)) \cup (\nabla S \bowtie T))</td>
</tr>
<tr>
<td>(S _transform T)</td>
<td>((\nabla S \bowtie T) \cup (S \bowtie \nabla T) \cup (((S - \nabla S) \bowtie (\Delta T \bowtie T)) \times {d_T}))</td>
</tr>
<tr>
<td>(S \times T)</td>
<td>((S \bowtie \nabla T) \cup (\nabla S \bowtie T) \cup (d_S) \times ((T - \nabla T) \bowtie (\Delta S \bowtie S))))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q</th>
<th>(\Delta Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S \bowtie T)</td>
<td>(((S - \nabla S) \bowtie (\Delta T \bowtie T)) \cup (\Delta S \bowtie T^m))</td>
</tr>
<tr>
<td>(S \sqcap T)</td>
<td>(((S - \nabla S) \bowtie (\nabla T \bowtie T^m)) \cup (\Delta S \bowtie T^m))</td>
</tr>
<tr>
<td>(S _transform T)</td>
<td>((\Delta S \bowtie T^m) \cup (\Delta T \bowtie T) \cup (((S - \nabla S) \bowtie (\nabla T \bowtie T^m)) \times {d_T}))</td>
</tr>
<tr>
<td>(S \times T)</td>
<td>((S \bowtie \Delta T) \cup (\Delta S \bowtie T^m) \cup (d_S) \times ((T - \nabla T) \bowtie (\Delta S \bowtie S^m))))</td>
</tr>
</tbody>
</table>

T. Griffin, B. Kumar: Algebraic Change Propagation for Semijoin and Outerjoin Queries. SIGMOD Record 1998
EXAMPLE QUERY #1

MATCH (v1)-[:a]-(v2)-[:b]-(v3)-[:c]-(v4)  ▪ a(v₁, v₂)
RETURNS v₁, v₂, v₃, v₄

Relational algebra expression: a ⨝ b ⨝ c

\[ \Delta(a ⨝ b ⨝ c) \]

\[ = (a ⨝ b)^m ⨝ \Delta c + (\Delta(a ⨝ b) ⨝ c^m) \]

\[ = (a ⨝ b)^m ⨝ \Delta c + (a^m ⨝ \Delta b ⨝ c^m) + (\Delta a ⨝ b^m ⨝ c^m) \]

\[ = (a^m ⨝ b^m ⨝ \Delta c) + (a^m ⨝ \Delta b ⨝ c^m) + (\Delta a ⨝ b^m ⨝ c^m) \]

Similarly to ∇(a ⨝ b ⨝ c).
EXAMPLE QUERY #1

MATCH (v1)-[:a]->(v2)-[:b]->(v3)-[:c]->(v4)
RETURN v1, v2, v3, v4

Δ(Δ(a ◊ b ◊ c))
= (a^n ◊ b^n ◊ Δc) + (a^n ◊ Δb ◊ c^m) + (Δa ◊ b^m ◊ c^m)

UNWIND $pcs AS pc
MATCH (v1)-[:a]->(v2)-[:b]->(v3)-[:pc]->(v4)
RETURN v1, v2, v3, v4

$pcs -> pass lists of nodes/edges as parameters
// This only works in embedded mode, see neo4j/issues/10239
POSITIVE DELTA QUERY FOR $a \bowtie b \bowtie c$

// r1 = a\bowtie b\bowtie \Delta c
UNWIND $pcs$ AS pc
MATCH (v1)-[:a]->(v2)-[:b]->(v3)-[pc]->(v4)
RETURN v1, v2, v3, v4
UNION ALL

// r2 = a\bowtie \Delta b\bowtie c
UNWIND $pbs$ AS pb
MATCH (v1)-[:a]->(v2)-[pb]->(v3)-[:c]->(v4)
RETURN v1, v2, v3, v4
UNION ALL

// r3 = \Delta a\bowtie b\bowtie c
UNWIND $pas$ AS pa
MATCH (v1)-[pa]->(v2)-[:b]->(v3)-[:c]->(v4)
RETURN v1, v2, v3, v4
POSITIVE DELTA QUERY FOR $a \bowtie b \bowtie c$

Long WITH chains are cumbersome -> patterns+list comprehensions.

WITH [pc IN $pcs | ((v1)-[:a]-(v2)-[:b]-(v3)-[pc]-(v4) | [v1, v2, v3, v4])] [pb IN $pbs | ((v1)-[:a]-(v2)-[pb]-(v3)-[:c]-(v4) | [v1, v2, v3, v4])] + [pa IN $pas | ((v1)-[pa]-(v2)-[:b]-(v3)-[:c]-(v4) | [v1, v2, v3, v4])] AS r

RETURN
EXAMPLE QUERY #2

MATCH (route:Route)
  -[:FOLLOWS]->(swP:SwitchPosition)
  -[:TARGET]->(sw:Switch)
  -[:MONITORED_BY]->(sensor:Sensor)
WHERE NOT (route)-[:REQUIRES]->(sensor)
RETURN route, sensor, swP, sw

MATCH (v1)
  -[:a]->(v2)
  -[:b]->(v3)
  -[:c]->(v4)
WHERE NOT (v1)-[:d]->(v4)
RETURN v1, v2, v3, v4
NEGATIVE CONDITIONS

MATCH (v1)
  -[:a]->(v2)
  -[:b]->(v3)
  -[:c]->(v4)
WHERE NOT (v1)-[:d]->(v4)
RETURN v1, v2, v3, v4

⇒ a ⋈ b ⋈ c ⋈ d
DELTA QUERIES FOR JOINS AND ANTIJOINS

Natural join

- $\Delta(S \bowtie T) = (\Delta S \bowtie T_m) + (S^m \bowtie \Delta T)$
- $\nabla(S \bowtie T) = (\nabla S \bowtie T) + (S \bowtie \nabla T)$

Antijoin

- $\Delta(S \bowtie T) = ((S - \nabla S) \bowtie (\nabla T \bowtie T^m)) + (\Delta S \bowtie T^m)$
- $\nabla(S \bowtie T) = ((S - \nabla S) \bowtie (\Delta T \bowtie T)) + (\nabla S \bowtie T)$

Only $S^m$ and $T^m$ are available.
DELTAS FOR ANTIJOINS

Based on Griffin-Kumar’s ’98 paper.

\[
\Delta(S \bar{\times} T) = ((S - \forall S) \times (\forall T \bar{\times} T^m)) + (\Delta S \bar{\times} T^m)
\]

\[
\Delta(S \bar{\times} T)^* = ((S^m - \Delta S) \times (\forall T \bar{\times} T^m)) + (\Delta S \bar{\times} T^m)
\]

\[
\Delta(S \bar{\times} T) = ((S^m - \Delta S) \times (\forall T \bar{\times} T^m)) + (\Delta S \bar{\times} T^m)
\]

\[
\forall(S \bar{\times} T) = ((S - \forall S) \times (\Delta T \bar{\times} T)) + (\forall S \bar{\times} T)
\]

\[
\forall(S \bar{\times} T)^* = ((S^m - \Delta S) \times (\Delta T \bar{\times} T)) + (\forall S \bar{\times} (T^m - \forall T))
\]

\[
\forall(S \bar{\times} T) = ((S^m - \Delta S) \times (\Delta T \bar{\times} T)) + (\forall S \bar{\times} (T^m - \forall T))
\]

\[
R^m = R - \forall R + \Delta R \quad \Rightarrow \quad * R = R^m - \Delta R + \forall R
\]
DELTA FOR ANTIJOINS

\[ \Delta((a \bowtie b \bowtie c) \bowtie d) = ((a \bowtie b \bowtie c)^m - \Delta(a \bowtie b \bowtie c)) \bowtie (\nabla d \bowtie d^m) + \Delta(a \bowtie b \bowtie c) \bowtie d^m \]

\[ = ((a \bowtie b \bowtie c)^m \bowtie (\nabla d \bowtie d^m)) - (\Delta(a \bowtie b \bowtie c) \bowtie (\nabla d \bowtie d^m)) + \Delta(a \bowtie b \bowtie c) \bowtie d^m \]

\[ = ((a \bowtie b \bowtie c)^m \bowtie (\nabla d \bowtie d^m)) - \left( (a^m \bowtie b^m \bowtie \Delta c) + (a^m \bowtie \Delta b \bowtie c^m) + (\Delta a \bowtie b^m \bowtie c^m) \right) \bowtie (\nabla d \bowtie d^m) \]

\[ + \Delta(a \bowtie b \bowtie c) \bowtie d^m \]

- Gets large very soon - workaround:
  - A-priori knowledge on the modifications helps reduce the scope
  - E.g. only consider changes in \(\Delta d\) and \(\nabla d\)
DELTAS FOR ANTIJOINS
LIST OPERATIONS IN CYPHER

Instead of subqueries, use chained queries and combine lists.

WITH [1, 2, 3] AS xs, [2] AS ys
RETURN
  xs + ys AS append,
  [x IN xs WHERE NOT x IN ys] AS subtraction,
  [x IN xs WHERE x IN ys] AS intersection

WITH [1, 1, 2, 2, 3] AS xs
RETURN
  reduce(acc = [], x in xs |
  acc + CASE x IN acc
  WHEN false THEN [x]
  ELSE []
  END) AS unique

Get unique list in openCypher

WITH [1, 1, 2, 2, 3] AS xs
UNWIND xs AS x
RETURN
  collect(DISTINCT x) AS unique
DELTA QUERIES IN CYPHER

Delta queries are complex. Features that would be nice:

- Subqueries //pattern comprehensions go some length
- Named subqueries //help reusability
- Subtracting lists //related: CIR-2017-180
- Use collection elements or function results for matching

MATCH (n)
WITH collect(n) AS ns
MATCH (ns[0])
RETURN *

MATCH (n)
WITH collect(n) AS ns
MATCH (ns[0]) AS x
MATCH (x)
RETURN *

These are probably too much to ask.

-> recommended approach: compile directly to query plans.
CHALLENGES FOR PROPERTY GRAPH QUERIES

Data model
- NF2 (Non-First Normal Form): maps, lists
- No schema (schema-optionality)
- Graph structure

Queries
- Nulls, antijoins and left outerjoins
- Updates on property values
- Aggregates on aggregates, non-distributive functions
- Ordering + skip/limit
- Reachability queries
# Challenges for Property Graph Queries

**Data model**
- NF2
- No schema
- Graph

**Queries**
- Nulls
- Updates
- Aggregates
- Ordering
- Reachability

<table>
<thead>
<tr>
<th>Decades of research</th>
<th>A. Gupta, I. S. Mumick: Materialized Views. MIT Press, 1999</th>
</tr>
</thead>
</table>
## Our Survey of Related IVM Techniques

<table>
<thead>
<tr>
<th>venue</th>
<th>contributions</th>
<th>A/P</th>
<th>bag</th>
<th>NF2</th>
<th>null</th>
<th>aggr.</th>
<th>ord.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGMOD’86</td>
<td>determining irrelevant updates, maintenance of SPJ views</td>
<td>A+P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TKDE’91</td>
<td>change propagation equations for relational alg.; fixed in [20]</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGMOD’93</td>
<td>counting algorithm (non-rec. views), DRed algorithm (rec. views)</td>
<td>P</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGMOD’96</td>
<td>change propagation equations for bag alg., incl. aggregation but no group-by</td>
<td>A</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBPL’97</td>
<td>extending IVM techniques to maintain views defined over a nested data model</td>
<td>A</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBPL’97</td>
<td>maintaining the transitive closure of directed graphs using a SQL-like language</td>
<td>A</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGMOD’97</td>
<td>group-by-aggregation, summary-deltas for representing changes</td>
<td>P</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TKDE’97</td>
<td>improved change propagation equations for relational algebra</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGMOD Record’98</td>
<td>change propagation equations for semijoins, antijoins and outer joins</td>
<td>A</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDEAS’99</td>
<td>incremental equations for the operators of the nested model</td>
<td>A</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLDB’02</td>
<td>maintenance of non-distributive aggregate functions</td>
<td>P</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>ER’03</td>
<td>order-preserving maintenance of XQuery views</td>
<td>A</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS’06</td>
<td>generalised summary-deltas, group-by-aggregations, outer joins; fixed in [35]</td>
<td>A</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICDE’03</td>
<td>top-k views</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICDE’07</td>
<td>outer joins and aggregation</td>
<td>P</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLDBJ’14</td>
<td>higher-order IVM, viewlet transformations, the DBToaster system</td>
<td>A</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overview of related literature on IVM techniques, presented in order of appearance. A/P: algebraic/procedural.
DBTOASTER

- Shows the scale of the problem
- Relational data model and SQL queries
- R&D for ~5 years @ EPFL, Johns Hopkins, Cornell, etc.
- Approach
  - Queries over an algebraic ring
  - Higher-order recursive IVM
- Compiler in OCaml
- Backend with code generation for C++, Scala/Spark

Christoph Koch et al.: DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views. VLDB Journal 2014
FUTURE DIRECTIONS

- Work out derivation rules for Expand/VarExpand, ...
- Automate delta query derivation
- Integrate to Neo4j
- Run performance experiments
  - Train Benchmark (set semantics)
  - LDBC Social Network Benchmark’s BI workload (bag semantics)
Short news
LDBC BENCHMARKS

- Social Network Benchmark
  - Business Intelligence workload published
  - openCypher reference implementation
  - Next goal: full conference paper

- Graphalytics
  - Competition is online at graphalytics.org
  - Neo4j implementation (using the Graph Algorithms library) WIP

Gábor Szárnyas, Arnau Prat-Pérez, Alex Averbuch, József Marton et al.: An early look at the LDBC Social Network Benchmark’s BI Workload. GRADES-NDA at SIGMOD, 2018

ldbc/ldbc_snb_implementations graphalytics-platforms-neo4j/pull/6
GRAPH ANALYTICS ON THE PANAMA PAPERS

- Network science approach: multidimensional graph metrics from social network analysis, biology, physics, etc.
- Our work originally targeted software and system models.
- Progress in 2018
  - Q1: implemented adapters for Neo4j and CSV
  - Q2 goal: analyse Panama papers and using metrics

Gábor Szárnyas, Zsolt Kővári, Ágnes Salánki, Dániel Varró: Towards the Characterization of Realistic Models: Evaluation of Multidisciplinary Graph Metrics, MODELS 2016

ftsrg/model-analyzer
MAPPING CYPHER TO SQL

- Evaluate graph queries in an RDB - similar to ORM
- Approaches
  - Cytosm: Cypher to SQL Mapper / gTop: graph topology
  - GraphGen - extracting graphs from RDBs
  - Ongoing work to map TCK to SQLite

B. A. Steer, A. Alnaimi, M. Lotz, F. Cuadrado, L. Vaquero, J. Varvenne: Cytosm: Declarative Property Graph Queries Without Data Migration. GRADES 2017

K. Xirogiannopoulos, V. Srinivas, A. Deshpande: GraphGen: Adaptive Graph Processing using Relational Databases. GRADES 2017
NEO4J APOC LIBRARY

- CSV loader that follows the schema of the neo4j-import tool
- Goal
  - Use headers to generate LOAD CSV commands.
  - 1\textsuperscript{st} pass: \texttt{CALL apoc.import.csv.node(file, labels, ...)}
  - 2\textsuperscript{nd} pass: \texttt{CALL apoc.import.csv.relationship(file, type, ...)}
- Result
  - Many corner cases -> ~700 LOC + tests
  - Covers most use cases, but is very slow
  - APOC PR pending

[neo4j-apoc-procedures/pull/581, neo4j-documentation/pull/121]
SCOPING FOR OPENCYPHERER

1. WITH 'dummy' AS x
2. MATCH (p:Person)-[:LIVES]->(c:City)
3. WITH p AS p, count(c) AS cities, x
4. MATCH (p:Person)-[:VISITED]->(c:Country)
5. WITH p, cities, count(c) AS countries, x
6. RETURN *

- Xtext grammar for the Sliza software analysis workbench
- Progress in 2018
  - Q1: scope analyser implemented for Cypher grammar of M05
  - Q2 goal: update to M10
Compiling openCypher graph queries with Spark Catalyst
Gábor Szárnyas

Graph-based analysis of JavaScript source code repositories
Gábor Szárnyas
Graph Processing devroom @ FOSDEM 2018

Writing a Cypher Engine in Clojure
Dávid Szakállas (BME)
Gábor Szárnyas (BME, MTA)

Learning Timed Automata with Cypher
Gábor Szárnyas, Rebeka Farkas, Márton Elekes, Anna Gujgiczer

3rd Budapest Clojure meetup 2018

Budapest Neo4j Meetup 13 February 2018