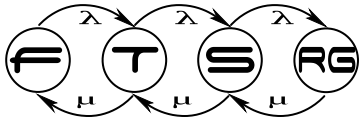
Budapest DB Meetup — 2018/Nov/13

Mapping Graph Queries to PostgreSQL

Gábor Szárnyas, József Marton,
Márton Elekes, János Benjamin Antal



Magyar Tudományos Akadémia

LDBC

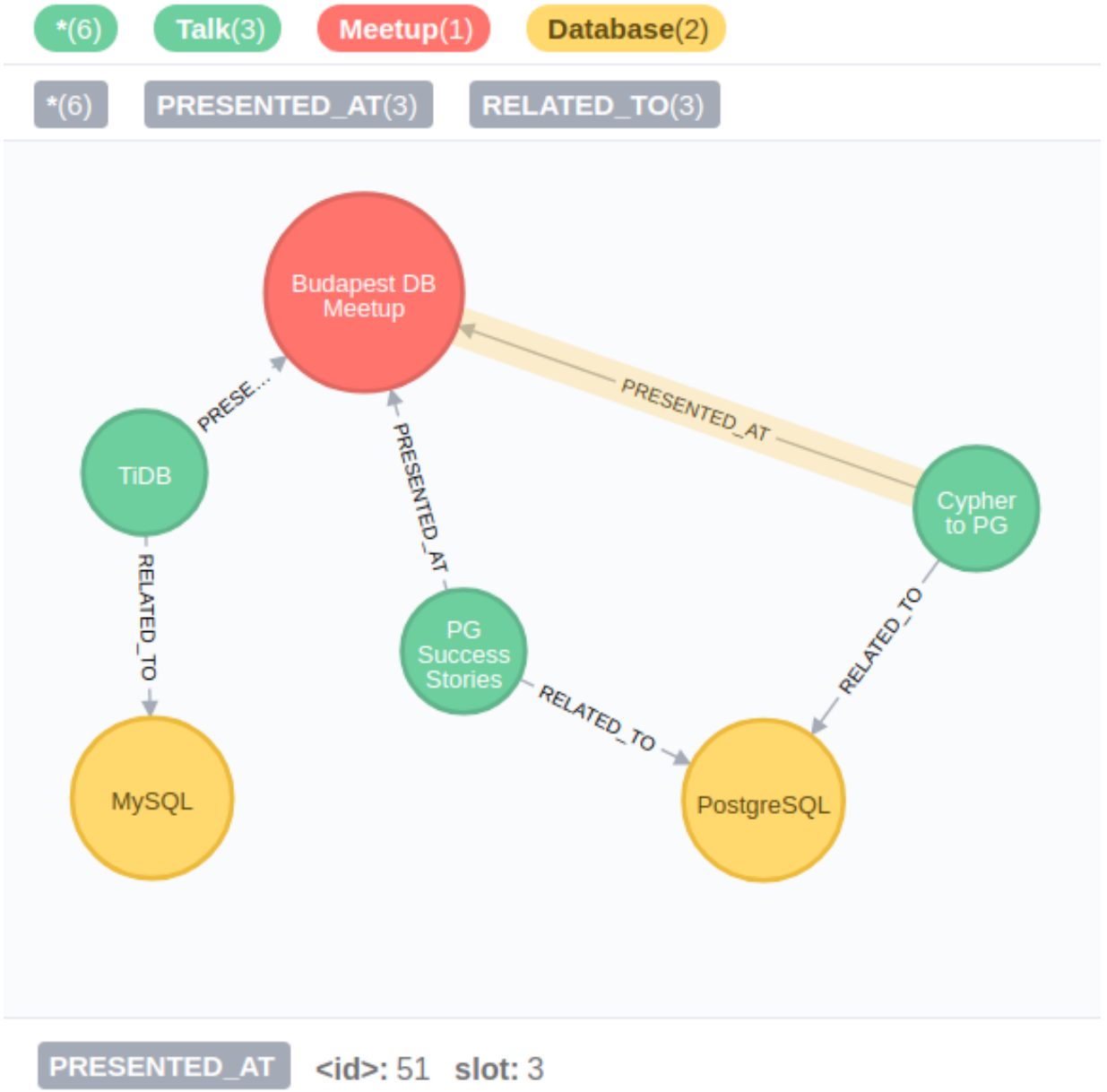
PROPERTY GRAPH DATABASES

NoSQL family

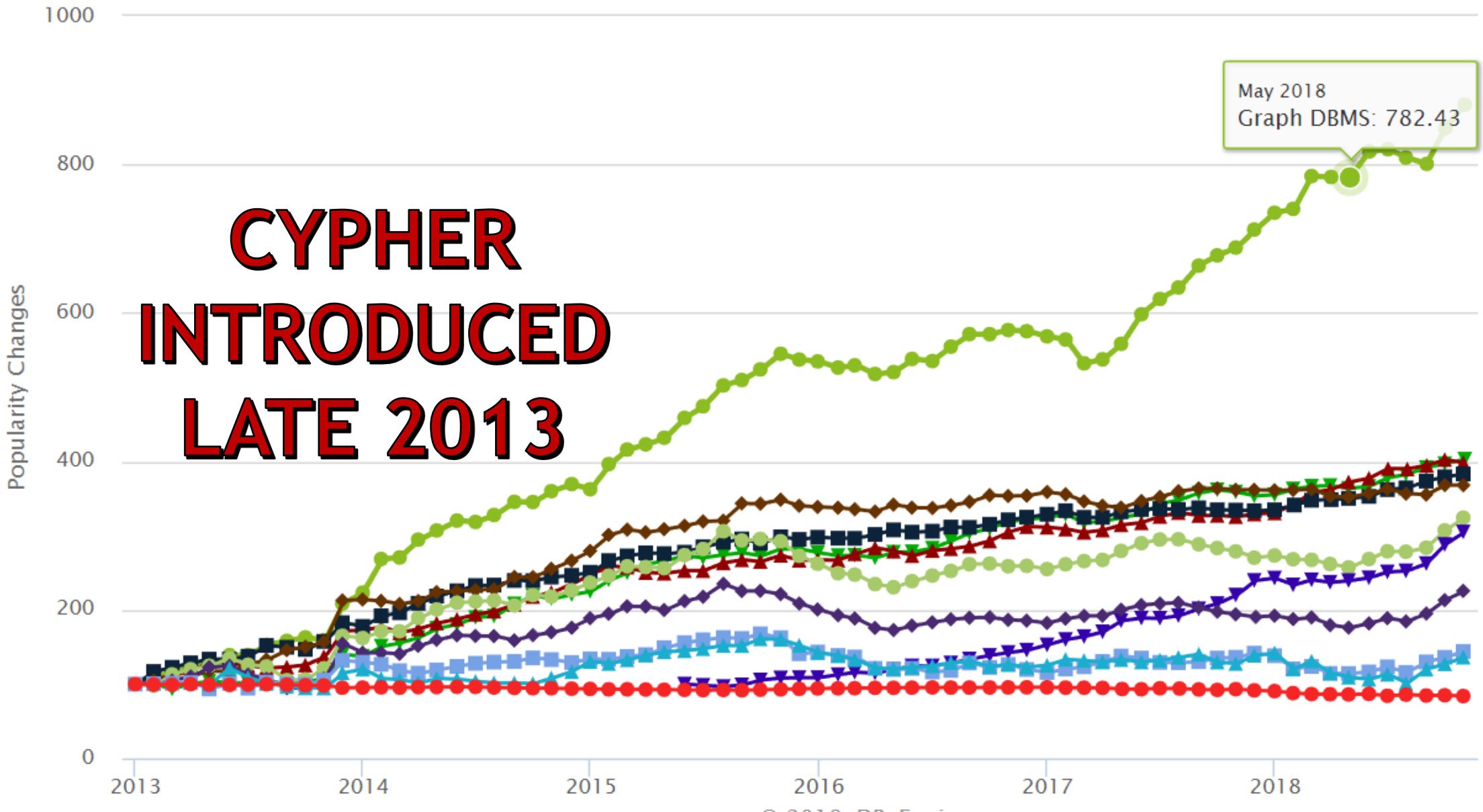
Data model:

- nodes
- edges
- properties

#1 query approach:
graph pattern matching



RANKINGS: POPULARITY CHANGES PER CATEGORY



CYPHER AND OPENCYPHER

Cypher: query language of the Neo4j graph database.

„Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax.”

```
MATCH
 (d:Database)<-[:RELATED_TO]-(:Talk)-[:PRESENTED_AT]->(m:Meetup)
WHERE m.date = 'Tuesday, November 13, 2018'
RETURN d
```

„The openCypher project aims to deliver a full and open specification of the industry’s most widely adopted graph database query language: Cypher.” (late 2015)

openCypher

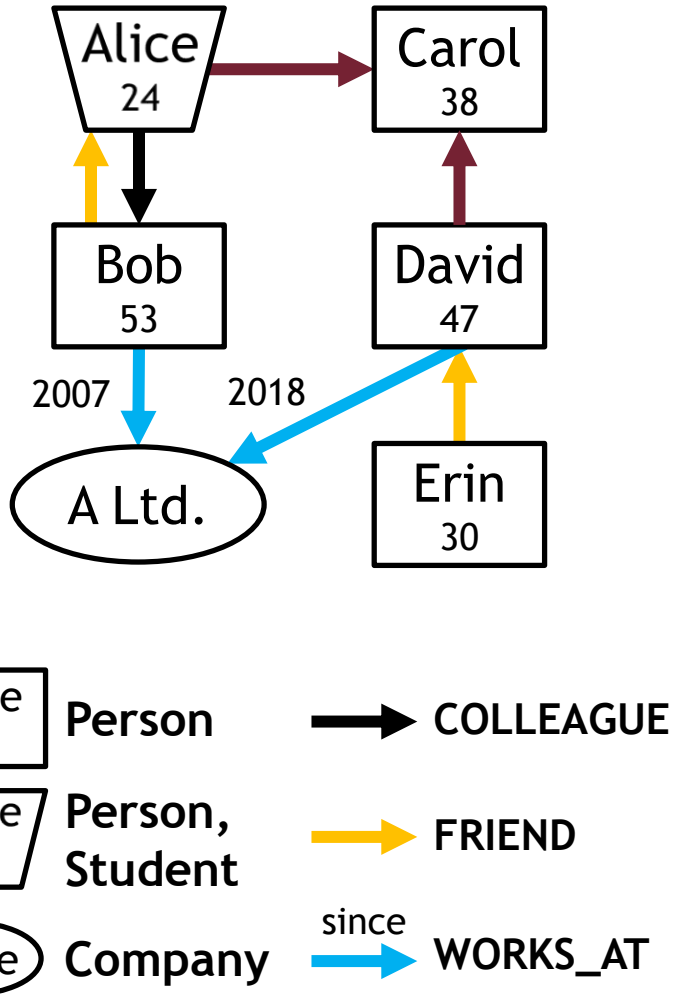
OPENCYPHER SYSTEMS

- Increasing adoption
- Relational databases:
  - SAP HANA
  - AGENS Graph
- Research prototypes:
  - Graphflow (Univesity of Waterloo, Canada)
  - ingraph (incremental graph engine @ BME)

(Source: Keynote talk @ GraphConnect NYC 2017)

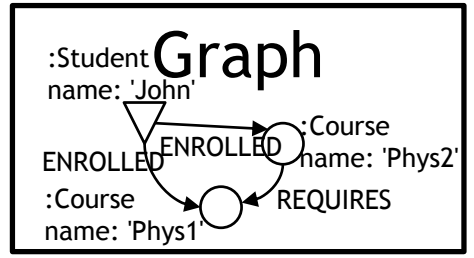
PROPERTY GRAPHS

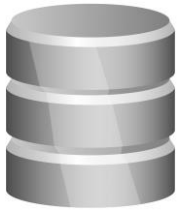
- Textbook graph: $G = (V, E)$
  - Dijkstra, Ford-Fulkerson, etc.
  - Homogenous nodes
  - Homogeneous edges
- Extensions
  - Labelled nodes
  - Typed edges
  - Properties
- The schema is implicit
- Very intuitive
  - *Things* and *connections*



GRAPH VS. RELATIONAL DATABASES

- Graph databases
  - Graph-based modelling is intuitive
  - Concise query language
- Relational databases
  - Most common
  - Many legacy systems
  - Efficient and mature
- No tools available to bridge the two
  - i.e. query data in RDBs as a graph
  - first you have to wrangle the graph out of the RDB





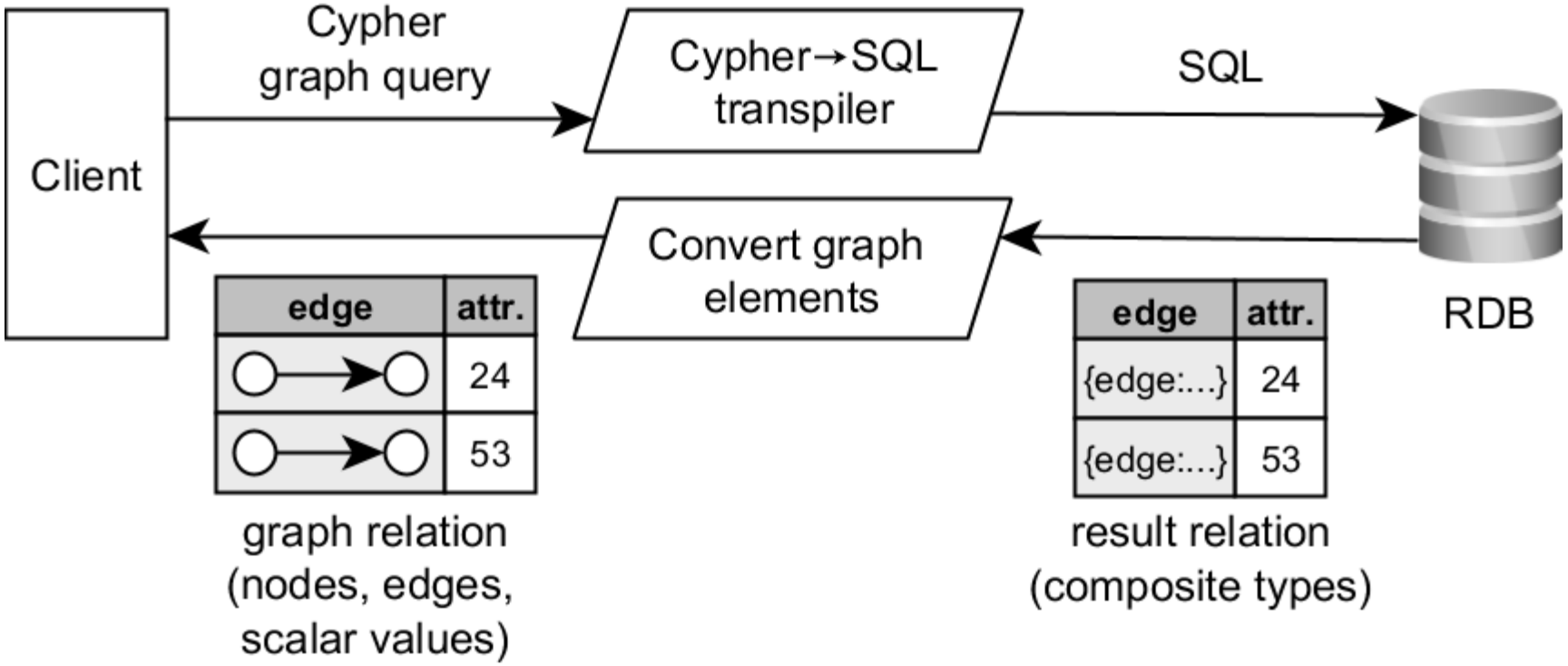
Tables

Col1	Col2
1	A
2	B

PROPOSED APPROACH

To get the best of both worlds, map Cypher queries to SQL:

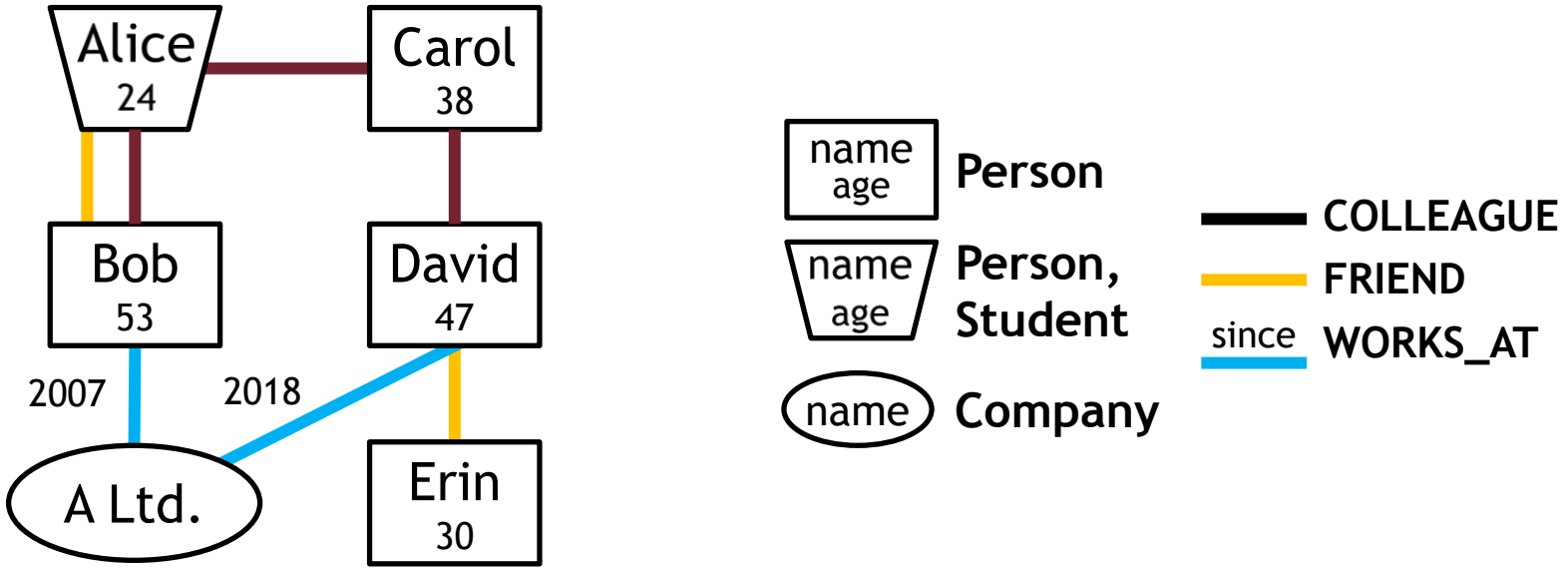
1. Formulate queries in Cypher
2. Execute inside an existing RDB
3. Return results as a graph relation



GRAPH QUERIES IN CYPHER

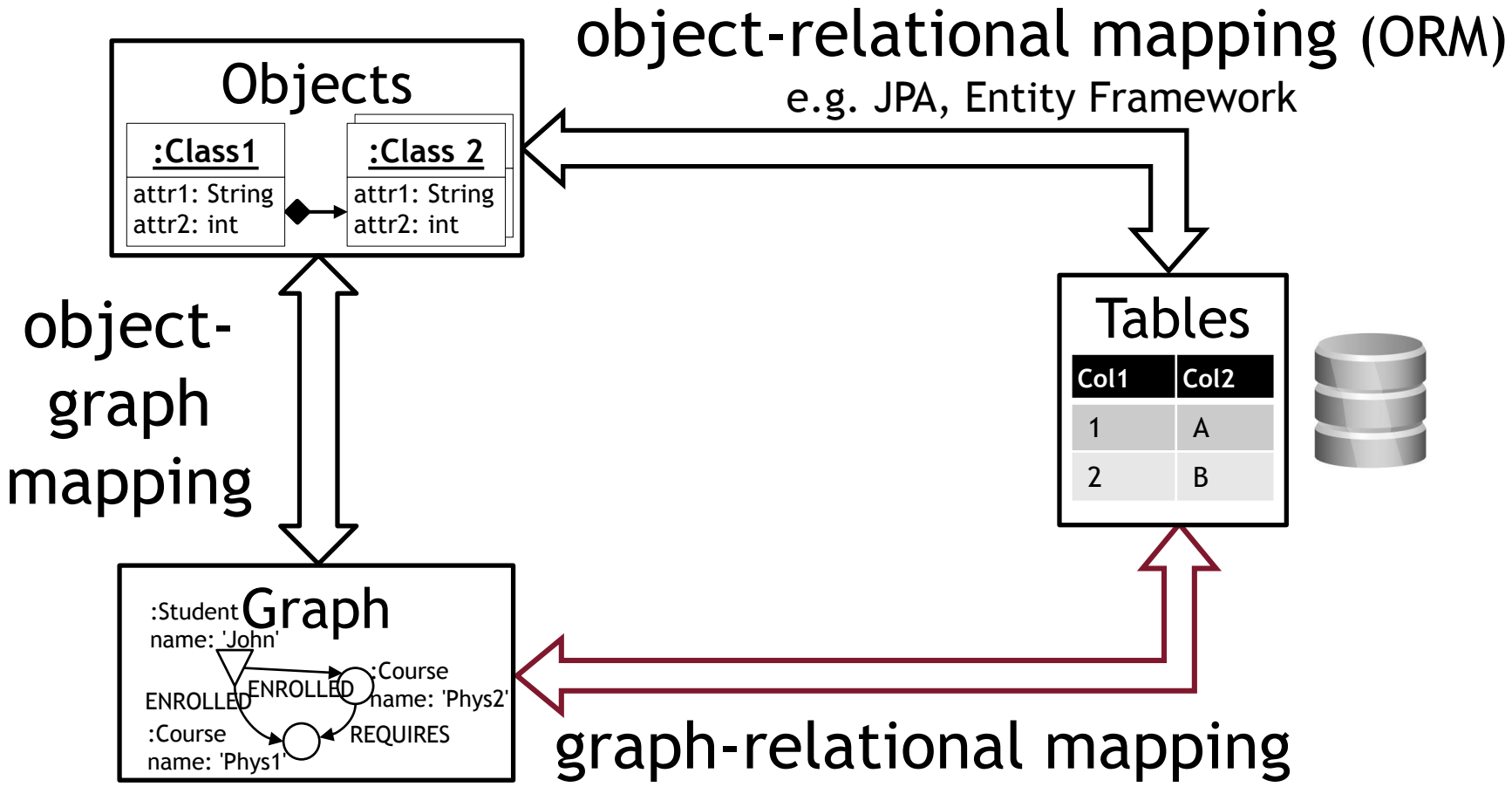
- Subgraph matching and graph traversals
- Example: Alice's colleagues and their colleagues

```
MATCH (p1:Person {name: 'Alice'})-[c:COLL*1..2]-(p2:Person)
RETURN p2.name
```



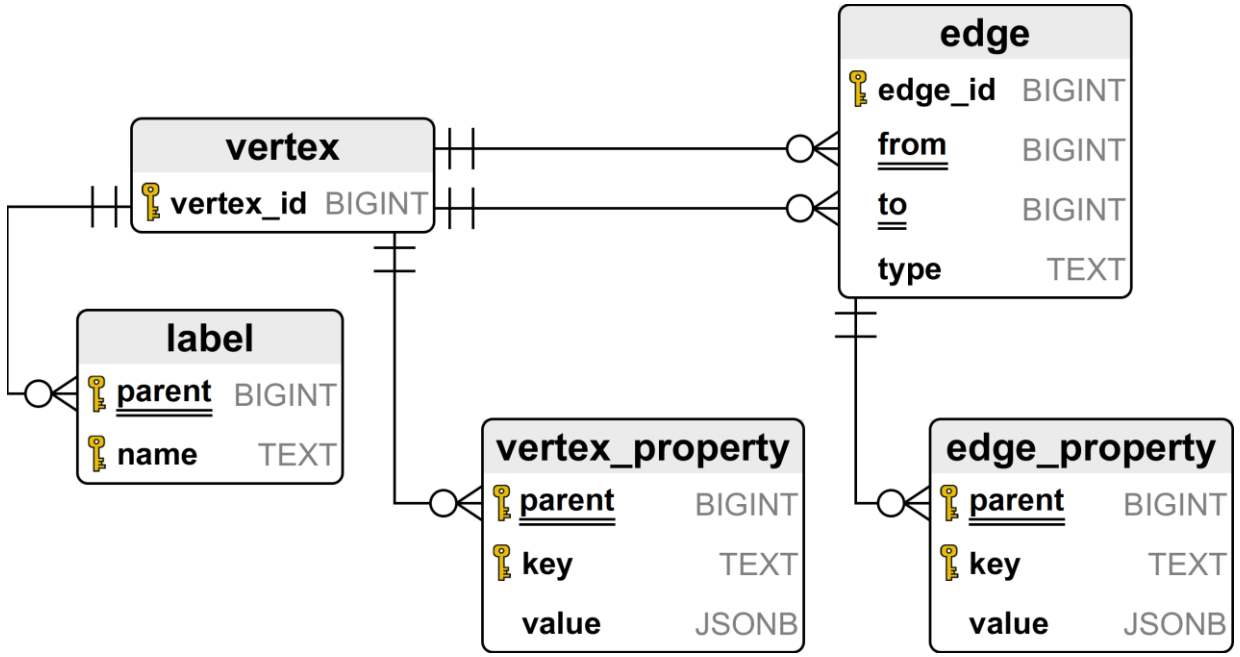
Data and Query Mapping

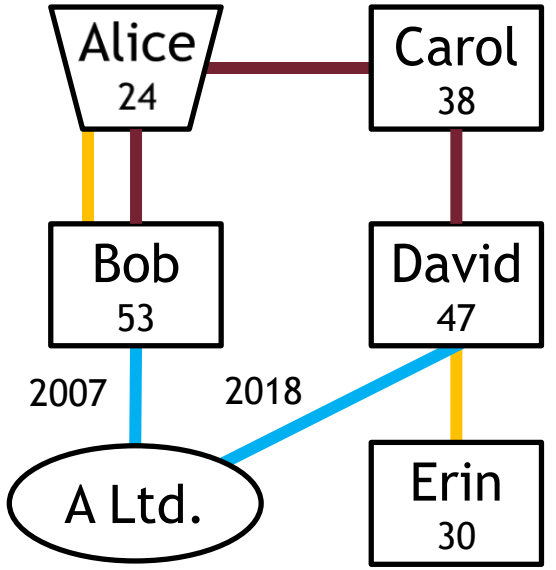
MAPPING BETWEEN DATA MODELS



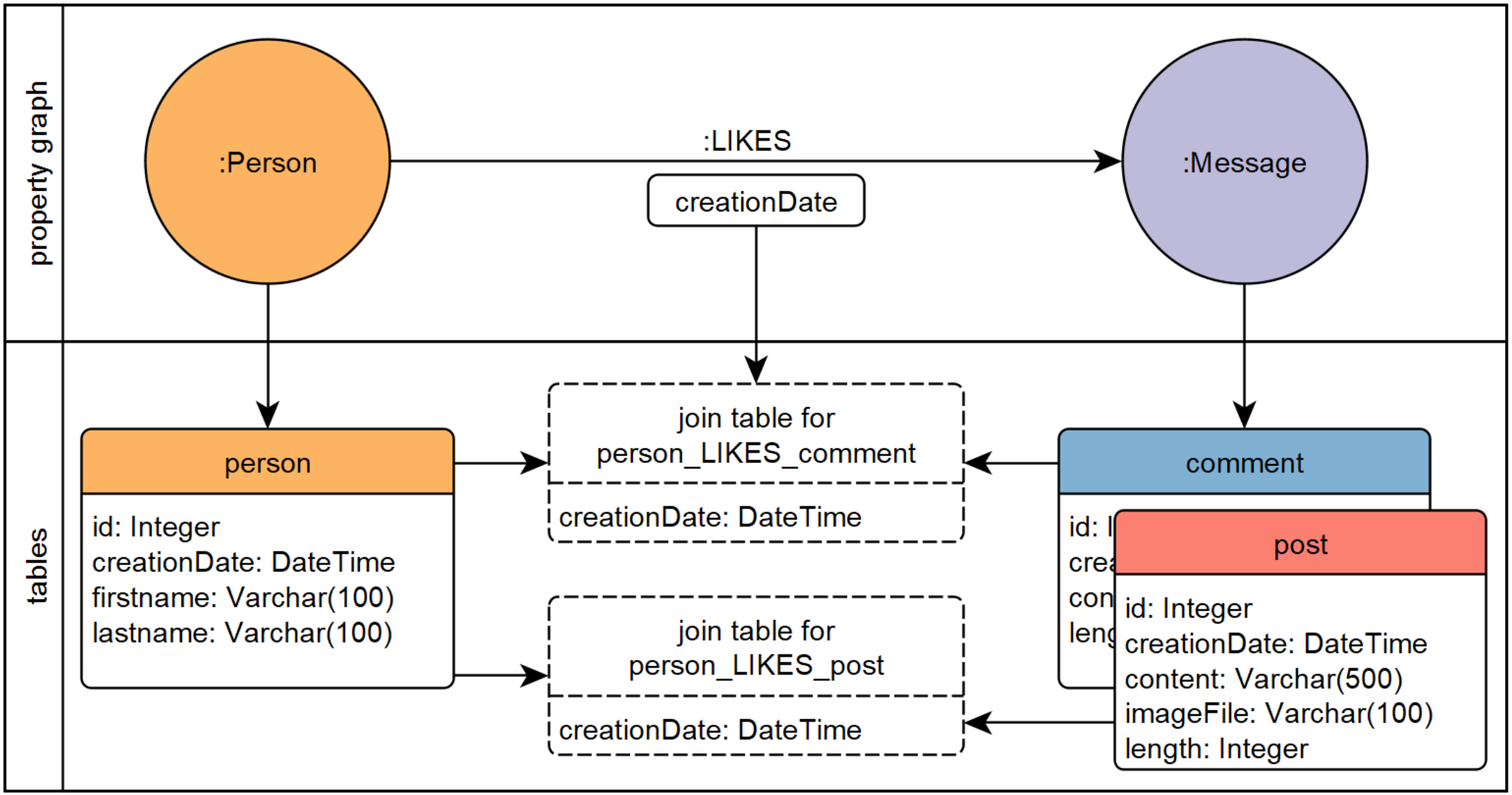
DATA MAPPING #1: GENERIC SCHEMA

- Useful for representing schema-free data sets
- Hopelessly slow

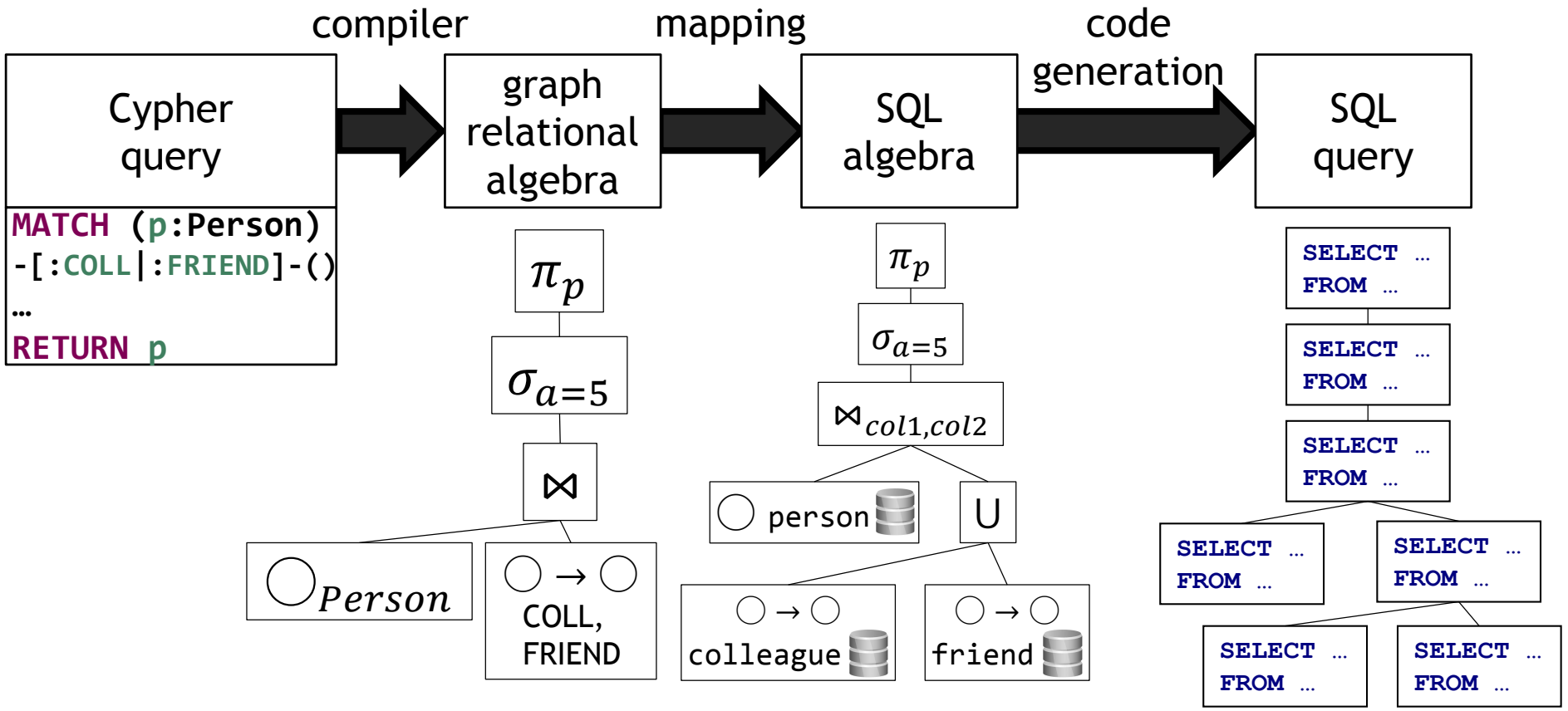




DATA MAPPING #2: CONCRETE SCHEMA



QUERY MAPPING



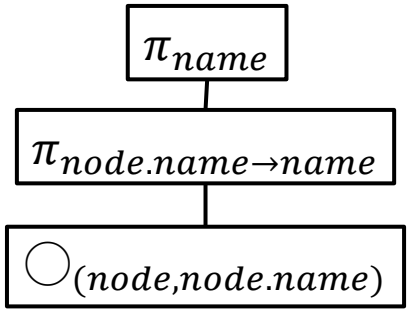
Gábor Szárnyas, József Marton, Dániel Varró:
Formalising openCypher Graph Queries in Relational Algebra.
ADBIS 2017

A SIMPLE EXAMPLE

```
MATCH (node)
WITH node.name AS name
RETURN name
```

Cypher query

ingraph mapping



relational algebra tree

```
SELECT "name"
FROM
  (
    SELECT "node.name" AS "name"
    FROM
      (
        SELECT
          vertex_id AS "node",
          (SELECT value
           FROM vertex_property
           WHERE parent = vertex_id AND key = 'name') AS "node.name"
        FROM vertex
      )
  )
```

CHALLENGES #1

- Variable length paths: union of multiple subqueries

```
MATCH (p1:Person {name: 'Alice'})-[c:COLL*1..2]-(p2:Person)
RETURN p2.name
```

- Unbound: `WITH RECURSIVE` (fixpoint-based evaluation)

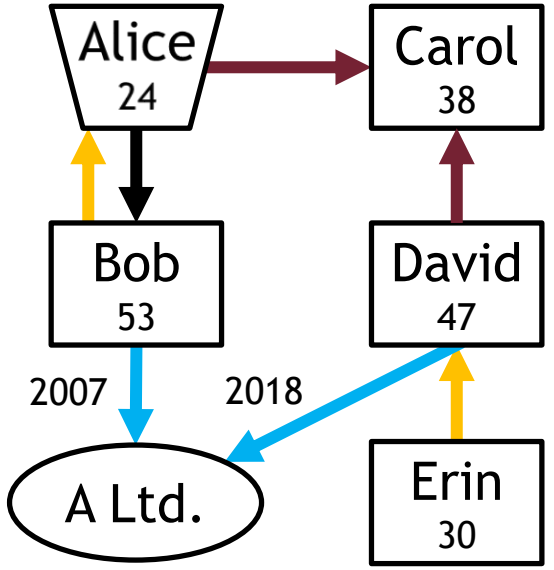
```
MATCH (p1:Person {name: 'Alice'})-[c:COLL*]-(p2:Person)
RETURN p2.name
```

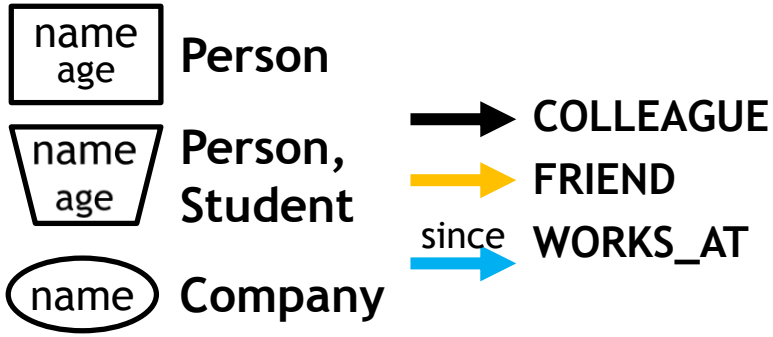
- `WITH RECURSIVE` was introduced in SQL:1999 but
  - PostgreSQL 8.4+ (since 2009)
  - SQLite 3.8.3+ (since 2014)
  - MySQL 8.0.1+ (since 2017)

CHALLENGES #2

- Edges are directed
  - Undirectedness is modelled in the query
  - Union of both directions

```
MATCH (p1:Person …)-[:FRIEND]-(p2:Person),
      (p2)-[:WORKS_AT]->(c:Company)
RETURN p2.name, c.name
```

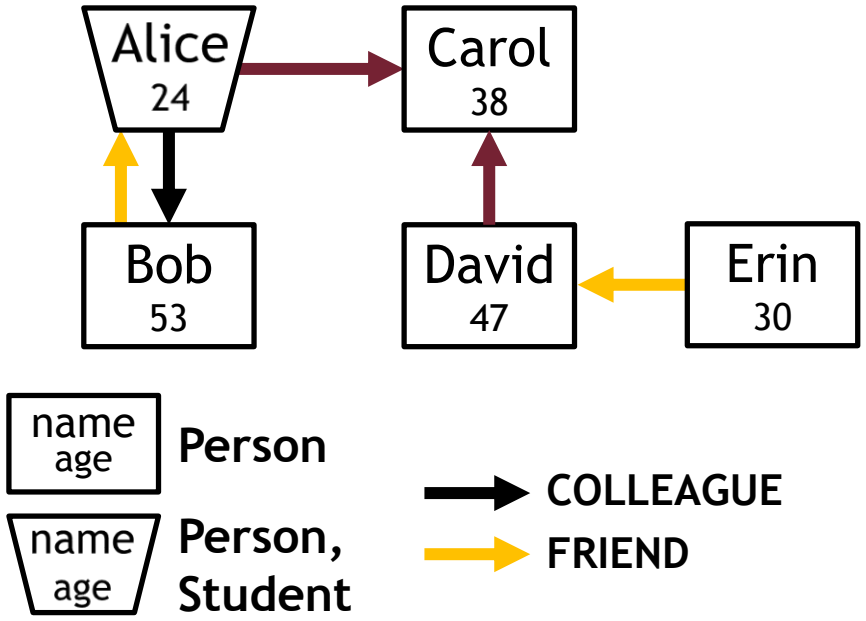


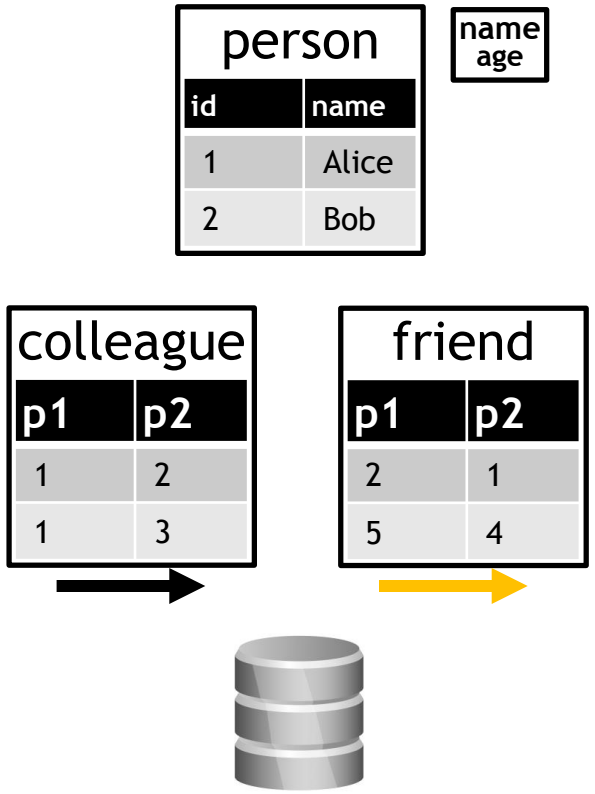


CHALLENGES #3

- Multiple tables as sources

```
MATCH (p1:Person …)-[:COLL|:FRIEND]-(p2:Person)
RETURN p2.name
```

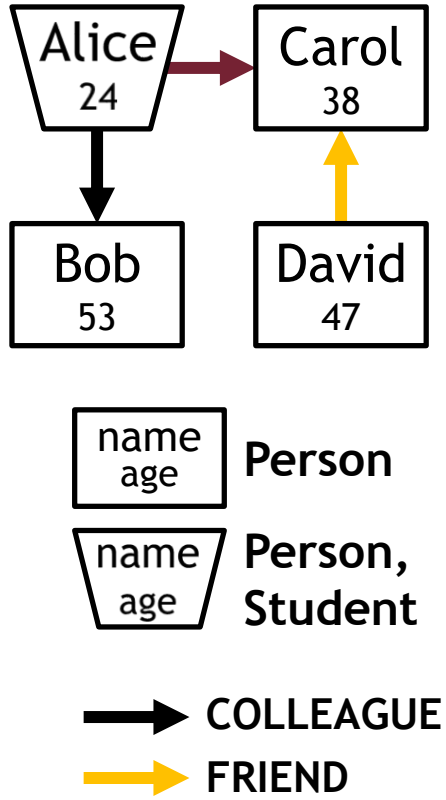




CHALLENGES #1 #2 #3

Simple graph patterns turn to many subqueries

- Querying edges as undirected:
  - Enumerate edges in both directions ($2\times$)
- Variable length paths (`1..L`, `*`):
  - Limited: enumerate $1, 2, \ldots, L \rightarrow L\times$
  - Unlimited: use `WITH RECURSIVE`
- Multiple node labels/edge types:
  - Enumerate all source tables ($N\times$)



Total: union of $2 \times L \times N$ subqueries in SQL

```
MATCH (p1:Person …)-[:COLL|:FRIEND*1..2]-(p2:Person)
```

A COMPLEX EXAMPLE

```
MATCH (:Person {id:$personId})-[:KNOWS]-(friend:Person)<-
[:HAS_CREATOR]-(message:Message)
WHERE message.creationDate <= $maxDate
RETURN
  friend.id AS personId,
  friend.firstName AS personFirstName,
  friend.lastName AS personLastName,
  message.id AS postOrCommentId,
  CASE exists(message.content)
    WHEN true THEN message.content
    ELSE message.imageFile
  END AS postOrCommentContent,
  message.creationDate AS postOrCommentCreationDate
ORDER BY postOrCommentCreationDate DESC, toInteger(postOrCommentId) ASC
LIMIT 20
```

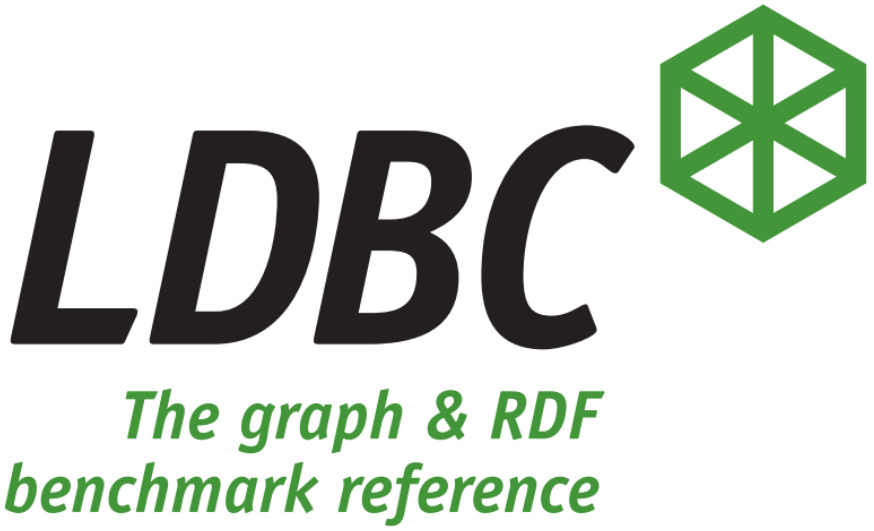
```
WITH
q0 AS
 (-- GetVerticesWithGTop
  SELECT
    ROW(0, p_personid)::vertex_type AS "_e186#0",
    "p_personid" AS "_e186.id#0"
  FROM person),
q1 AS
 (-- Selection
  SELECT * FROM q0 AS subquery
  WHERE ("_e186.id#0" = :personId)),
q2 AS
 (-- GetEdgesWithGTop
  SELECT ROW(0, edgeTable."k_person1id")::vertex_type AS "_e186#0", ROW(0, edgeTable."k_person1id", edgeTable."k_person2id")::edge_type AS
"_e187#0", ROW(0, edgeTable."k_person2id")::vertex_type AS "friend#2",
    toTable."p_personid" AS "friend.id#2", toTable."p_firstname" AS "friend.firstName#1", toTable."p_lastname" AS "friend.lastName#2"
    FROM "knows" edgeTable
      JOIN "person" toTable ON (edgeTable."k_person2id" = toTable."p_personid")),
q3 AS
 (-- GetEdgesWithGTop
  SELECT ROW(0, edgeTable."k_person1id")::vertex_type AS "friend#2", ROW(0, edgeTable."k_person1id", edgeTable."k_person2id")::edge_type AS
"_e187#0", ROW(0, edgeTable."k_person2id")::vertex_type AS "_e186#0",
    fromTable."p_personid" AS "friend.id#2", fromTable."p_firstname" AS "friend.firstName#1", fromTable."p_lastname" AS "friend.lastName#2"
    FROM "knows" edgeTable
      JOIN "person" fromTable ON (fromTable."p_personid" = edgeTable."k_person1id")),
q4 AS
 (-- UnionAll
  SELECT "_e186#0", "_e187#0", "friend#2", "friend.id#2", "friend.firstName#1", "friend.lastName#2" FROM q2
  UNION ALL
  SELECT "_e186#0", "_e187#0", "friend#2", "friend.id#2", "friend.firstName#1", "friend.lastName#2" FROM q3),
q5 AS
 (-- EquiJoinLike
  SELECT left_query."_e186#0", left_query."_e186.id#0", right_query."friend#2", right_query."friend.id#2", right_query."_e187#0",
right_query."friend.lastName#2", right_query."friend.firstName#1" FROM
    q1 AS left_query
    INNER JOIN
    q4 AS right_query
  ON left_query."_e186#0" = right_query."_e186#0"),
q6 AS
 (-- GetEdgesWithGTop
  SELECT ROW(6, fromTable."m_messageid")::vertex_type AS "message#17", ROW(8, fromTable."m_messageid", fromTable."m_creatorid")::edge_type AS
"_e188#0", ROW(0, fromTable."m_creatorid")::vertex_type AS "friend#2",
    fromTable."m_messageid" AS "message.id#2", fromTable."m_content" AS "message.content#2", fromTable."m_ps_imagefile" AS
"message.imageFile#0", fromTable."m_creationdate" AS "message.creationDate#13"
    FROM "message" fromTable
  WHERE (fromTable."m_c_replyof" IS NULL)),
q7 AS
 (-- GetEdgesWithGTop
  SELECT ROW(6, fromTable."m_messageid")::vertex_type AS "message#17", ROW(8, fromTable."m_messageid", fromTable."m_creatorid")::edge_type AS
"_e188#0", ROW(0, fromTable."m_creatorid")::vertex_type AS "friend#2",
    fromTable."m_messageid" AS "message.id#2", fromTable."m_content" AS "message.content#2", fromTable."m_creationdate" AS
"message.creationDate#13"
    FROM "message" fromTable
  WHERE (fromTable."m_c_replyof" IS NOT NULL)),
q8 AS
 (-- UnionAll
  SELECT "message#17", "_e188#0", "friend#2", "message.id#2", "message.content#2", "message.imageFile#0", "message.creationDate#13" FROM q6
  UNION ALL
  SELECT "message#17", "_e188#0", "friend#2", "message.id#2", "message.content#2", NULL AS "message.imageFile#0", "message.creationDate#13"
FROM q7),
q9 AS
 (-- EquiJoinLike
  SELECT left_query."_e186#0", left_query."_e186.id#0", left_query."_e187#0", left_query."friend#2", left_query."friend.id#2",
left_query."friend.firstName#1", left_query."friend.lastName#2", right_query."message#17", right_query."message.id#2",
right_query."message.imageFile#0", right_query."_e188#0", right_query."message.creationDate#13", right_query."message.content#2" FROM
    q5 AS left_query
    INNER JOIN
    q8 AS right_query
  ON left_query."friend#2" = right_query."friend#2"),
q10 AS
 (-- AllDifferent
  SELECT * FROM q9 AS subquery
    WHERE is_unique(ARRAY[]::edge_type[] || "_e188#0" || "_e187#0")),
q11 AS
 (-- Selection
  SELECT * FROM q10 AS subquery
  WHERE ("message.creationDate#13" <= :maxDate)),
q12 AS
 (-- Projection
  SELECT "friend.id#2" AS "personId#0", "friend.firstName#1" AS "personFirstName#0", "friend.lastName#2" AS "personLastName#0",
"message.id#2" AS "postOrCommentId#0", CASE WHEN ("message.content#2" IS NOT NULL = true) THEN "message.content#2"
           ELSE "message.imageFile#0"
         END AS "postOrCommentContent#0", "message.creationDate#13" AS "postOrCommentCreationDate#0"
    FROM q11 AS subquery),
q13 AS
 (-- SortAndTop
  SELECT * FROM q12 AS subquery
    ORDER BY "postOrCommentCreationDate#0" DESC NULLS LAST, ("postOrCommentId#0")::BIGINT ASC NULLS FIRST
    LIMIT 20)
SELECT "personId#0" AS "personId", "personFirstName#0" AS "personFirstName", "personLastName#0" AS "personLastName", "postOrCommentId#0" AS
"postOrCommentId", "postOrCommentContent#0" AS "postOrCommentContent", "postOrCommentCreationDate#0" AS "postOrCommentCreationDate"
  FROM q13 AS subquery
```

Benchmarks

BENCHMARKS: LINKED DATA BENCHMARK COUNCIL

LDBC is a non-profit organization dedicated to establishing benchmarks, benchmark practices and benchmark results for graph data management software.

The Social Network Benchmark is an industrial and academic initiative, formed by principal actors in the field of graph-like data management.



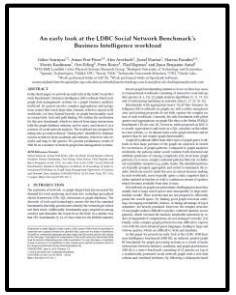
LDBC IN A NUTSHELL


TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark

Peter Boncz, Thomas Neumann, Orri Erling,
TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark,
TPCTC 2013


The LDBC Social Network Benchmark: Interactive Workload

Orri Erling et al.,
The LDBC Social Network Benchmark: Interactive Workload,
SIGMOD 2015

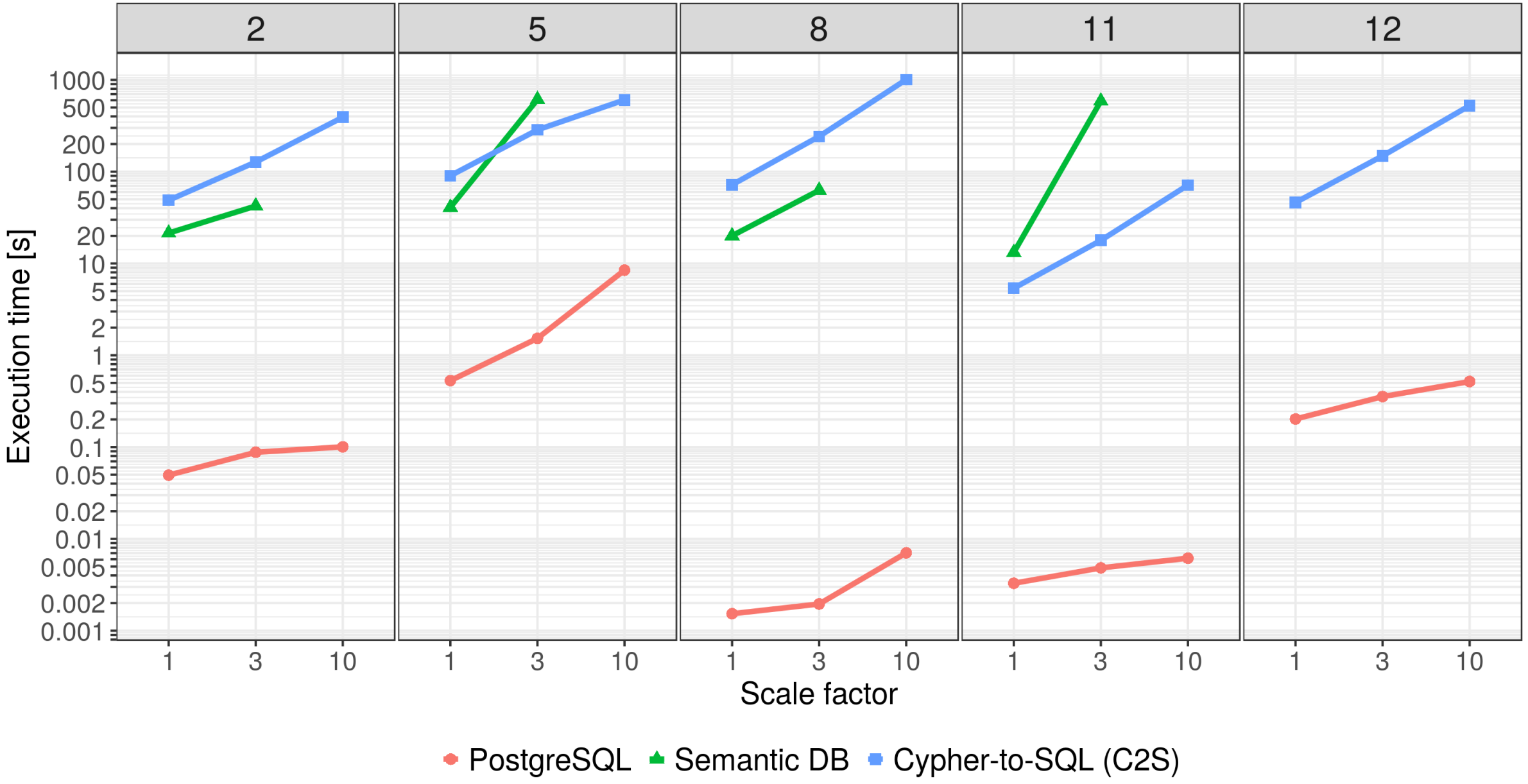

An early look at the LDBC Social Network Benchmark's Business Intelligence workload

Gábor Szárnyas, József Marton, János Benjamin Antal et al.:
An early look at the LDBC Social Network Benchmark's BI Workload.
GRADES-NDA at SIGMOD, 2018

PERFORMANCE EXPERIMENTS

- LDBC Interactive workload
- Tools
  - PostgreSQL (reference implementation)
  - Cypher-to-SQL queries on PostgreSQL
  - Semantic database (anonymized)
- Geometric mean of 20+ executions

BENCHMARK RESULTS ON LDBC QUERIES



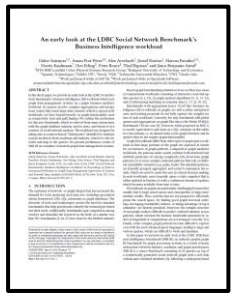
RELATED PROJECTS

- Cypher for Apache Spark
  - Neo4j's project
  - Executes queries in Spark
  - Read-only
- Cytosm
  - Cypher to SQL Mapping
  - HP Labs for Vertica
  - Project abandoned in 2017
  - gTop (graph topology) reused

Tool	Source	Target	OSS	Updates	Paths
CAPS	Cypher	SparkSQL	✓	✘	✘
Cytosm	Cypher	Vertica SQL	✓	✘	✘
Cypher-to-SQL	Cypher	PostgreSQL	✓	✓	✓

SUMMARY

- Mapping property graph queries to SQL is challenging
  - Similar to ORM
  - + edge properties
  - + reachability
- Initial implementation: C2S
  - Moderate feature coverage
  - Poor performance
  - Needs some tweaks, e.g. work around CTE optimization fences



Gábor Szárnyas, József Marton, János Maginecz, Dániel Varró: *Incremental View Maintenance on Property Graphs*. arXiv preprint 2018

RELATED RESOURCES

ingraph and C2S	github.com/ftsrg/ingraph
Cypher for Apache Spark	github.com/opencypher/cypher-for-apache-spark
Cytosm	github.com/cytosm/cytosm
LDBC	github.com/ldbc/

Thanks for the contributions to the whole ingraph team.