LDBC *The graph & RDF benchmark reference*

LDBC Graph Query Language Task Force

Status Report – 9th TUC Meeting

Speaker: Hannes Voigt, TU Dresden

Aim

- Study query languages for graph data management systems, specifically systems storing "Property Graph" data
- Query language should cover the needs of important use cases: social network benchmark, interactive and BI workloads

- Members
  - Renzo Angles, Universidad de Talca
  - **Marcelo Arenas, PUC Chile - task force lead**
  - Pablo Barceló, Universidad de Chile
  - Peter Boncz, Vrije Universiteit Amsterdam
  - George Fletcher, Eindhoven University of Technology
  - Claudio Gutierrez, Universidad de Chile
  - Tobias Lindaaker, Neo Technology
  - Marcus Paradies, SAP
  - Raquel Pau, UPC
  - Arnau Prat, UPC / Sparsity
  - Juan Sequeda, Capsenta
  - Oskar van Rest, Oracle Labs
  - Hannes Voigt, TU Dresden
  - Yinglong Xia, IBM

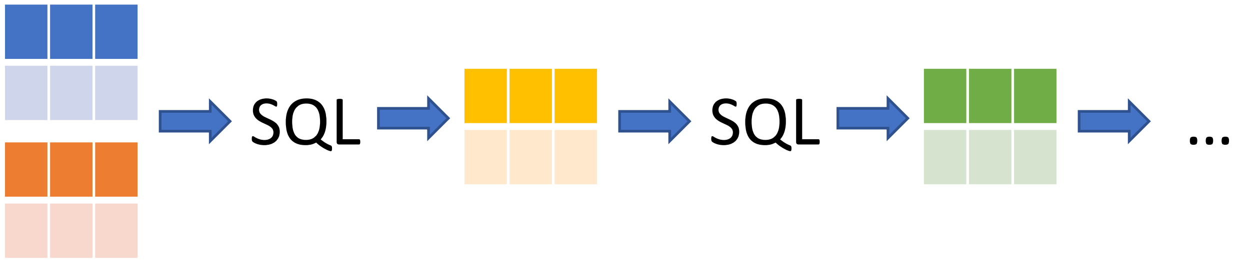
Motivation

- Currently Babel of graph QLs with diversity in syntax and semantics
  - PGQL: iso/homomorphism
  - Cypher: “edge” isomorphism
  - Gremlin: homomorphism
  - SPARQL: homomorphism
  - Reachability queries vs. path queries

[By Pieter Brueghel the Elder (1526/1530–1569)]

- Practical consequences
  - Applications are not portable
  - Hard to define benchmarks
  - Hard to compare Graph DBMS
- Standard
  - Prevents vendor lock-in
  - Fosters true performance competition → improvement of systems

Closed Query Languages

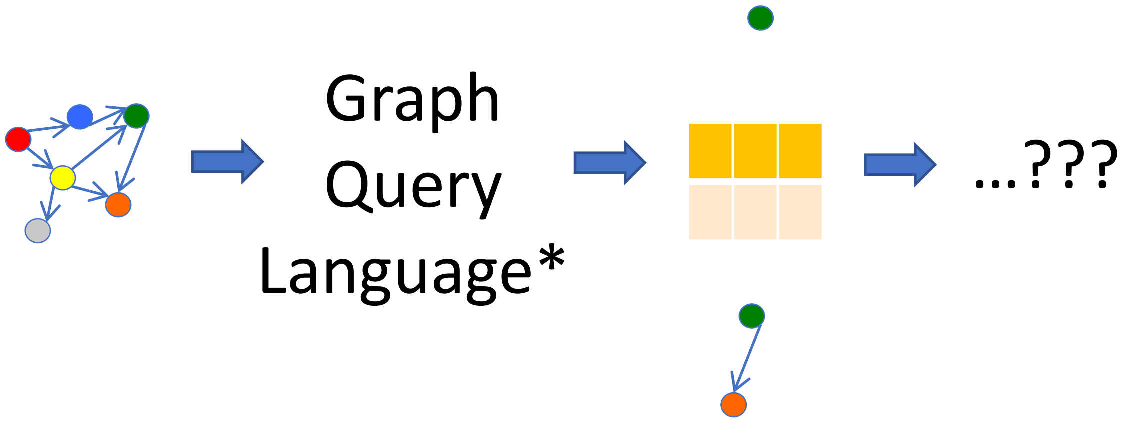


SQL

SQL

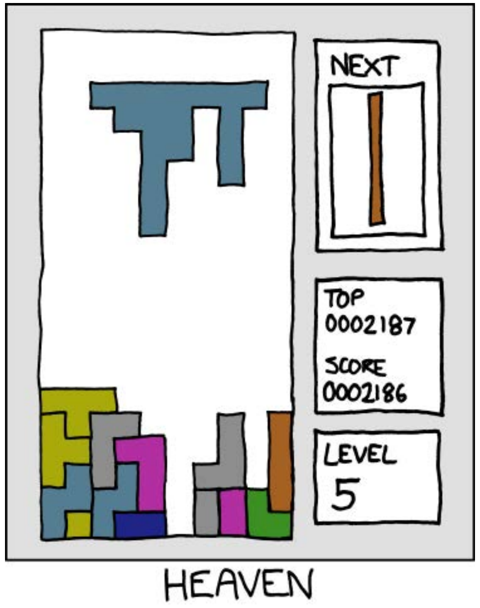
…

however …

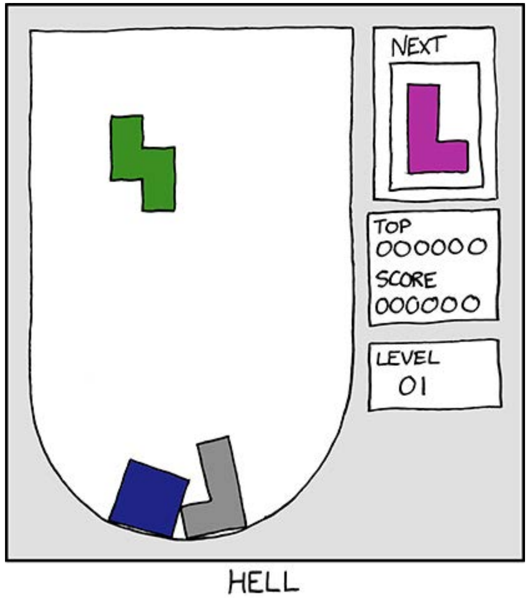


Graph Query Language*

…???



[xkcd, https://xkcd.com/888/]

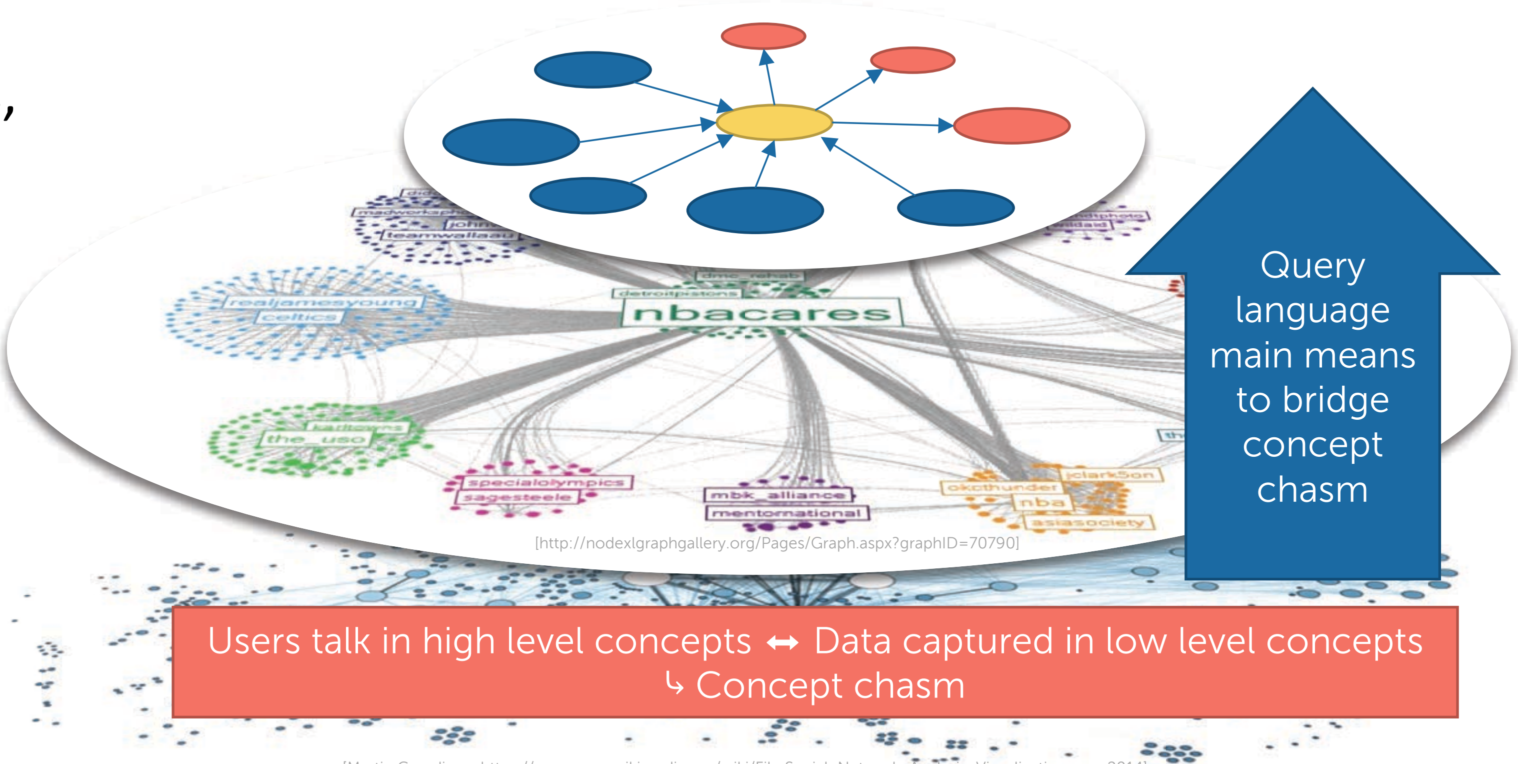


[xkcd, https://xkcd.com/724/]

Cross the Concept Chasm with Composability

- Users talk about…
  - Application entities
  - e.g. discussions, topics, communities, etc.
  - Likely multiple abstraction levels

- Base data contains…
  - Fine granular data
  - Low abstraction
  - E.g. individual twitter messages, retweet relationships, etc.



Query language main means to bridge concept chasm

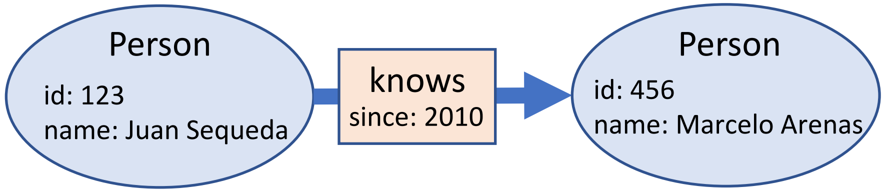
[http://nodexlgraphgallery.org/Pages/Graph.aspx?graphID=70790]

Users talk in high level concepts ↔ Data captured in low level concepts
↳ Concept chasm

[Martin Grandjean, https://commons.wikimedia.org/wiki/File:Social_Network_Analysis_Visualization.png, 2014]

High-level Design Goals

- Query Language for Property Graph Model
  - Power comparable to SQL 92

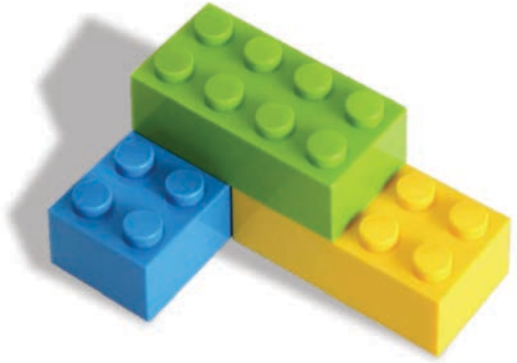


Person
id: 123
name: Juan Sequeda

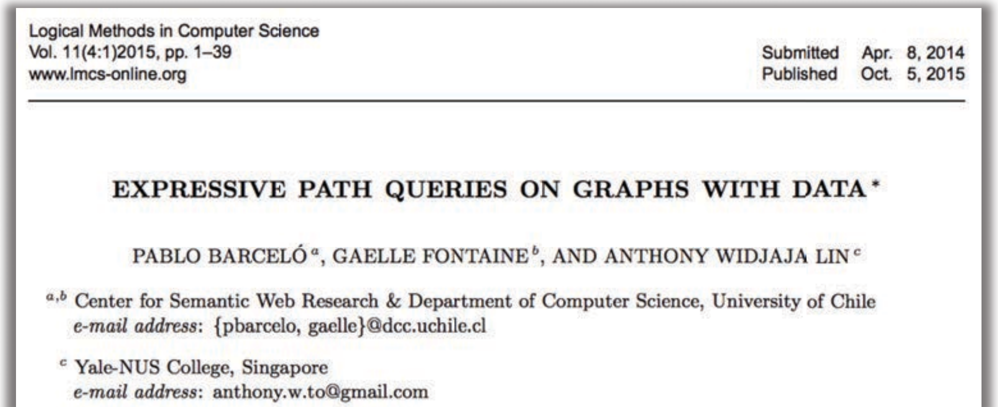
knows
since: 2010

Person
id: 456
name: Marcelo Arenas

- Composability (language closed over data model)
- Orthogonal language concepts



- Paths as first class citizen



Logical Methods in Computer Science
Vol. 11(4:1)2015, pp. 1–39
www.lmcs-online.org

Submitted Apr. 8, 2014
Published Oct. 5, 2015

EXPRESSIVE PATH QUERIES ON GRAPHS WITH DATA *

PABLO BARCELÓ [a], GAELLE FONTAINE [b], AND ANTHONY WIDJAJA LIN [c]

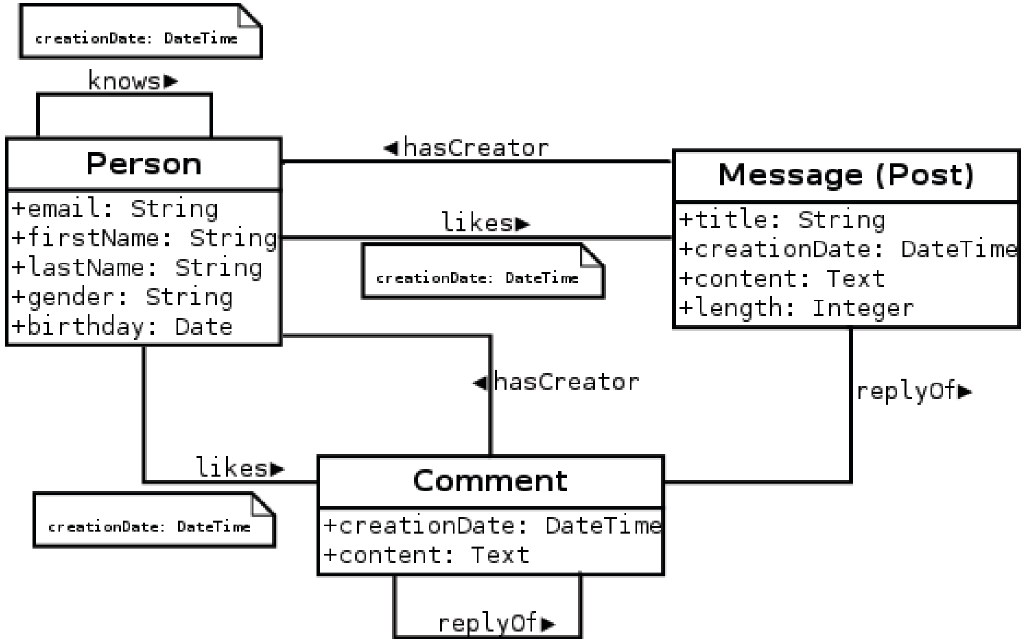
[a,b] Center for Semantic Web Research & Department of Computer Science, University of Chile
e-mail address: {pbarcelo, gaelle}@dcc.uchile.cl

[c] Yale-NUS College, Singapore
e-mail address: anthony.w.to@gmail.com

Where are we now?

Catalog of Desired Query Functionalities

- Adjacency Queries
- Graph Pattern Matching
- Navigational Queries
- Aggregate Queries
- Sub queries



Adjacency queries

- Property access
  - Get the firstName and lastName of a person having email "$email”
- Neighborhood of a node
  - Get the firstName and lastName of the friends of a person identified by email "$email".
- K-neighborhood of a node
  - Get the email, firstName and lastName of friends of the friends of a person having email "$email" (excluding the start person) (i.e. get a list of recommended friends) (directed 2-neighborhood)

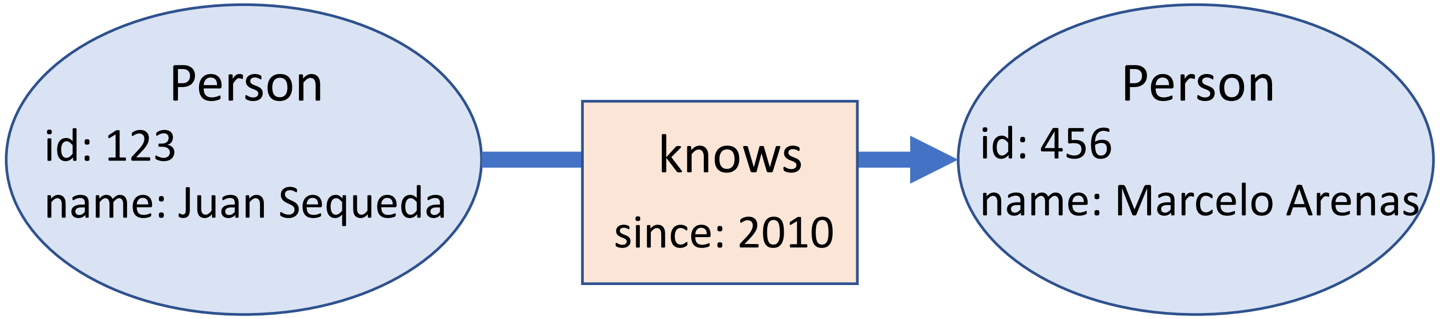
Graph Pattern Matching

- Join
  - Get the creationDate and content of the messages created by a person identified by email "$email1" and commented by another person identified by "$email2".
- Union
  - Get the creationDate and content of the messages either created or liked by a person identified by email "$email".
- Intersection
  - Get the email, firstName and lastName of the common friends between two persons identified by emails "$email1" and "$email2" respectively.
- Difference
  - Given two friends identified by emails "$email1" and "$email2" respectively, get the email, firstName and lastName of the friends of the second person which are not friends of the first person (this questions is relevant for friendship recommendations).
- Optional
  - Given a person identified by email "$email", get the title of all the messages created by such person, and the content of the first comment replying each message (if it exists).
- Filter
  - Get the properties of the people whose firstName includes the string "xxx" (it implies use of wildcards).

Navigational queries

- Reachability
  - Is there a friendship connection between two persons identified by emails "$email1" and "$email2" respectively?
- All Path Finding
  - Get the friendship paths between two persons identified by emails "$email1" and "$email2" respectively.
- Shortest Path Finding
  - The shortest friendship path between two persons identified by emails "$email1" and "$email2" respectively".
- Regular Path Query
  - Get the firstName of friends of the friends of the friends of a person identified by email "$email".
- Conjunctive Regular Path Queries
  - Given a target message created on "$dateTime" by a person identified by email "$email", for each comment replying the target message, get the comment's content and the email of the comment´s creator.
- Filtered regular path query
  - Given a person identified by email "$email", get the title of all the messages liked by such person between "$dateTime1" and "$dateTime2”.

Data Model


Person
id: 123
name: Juan Sequeda

knows
since: 2010

Person
id: 456
name: Marcelo Arenas

- **L** is an infinite set of (node and edge) labels;
- **K** is an infinite set of property names
- **V** is an infinite set of literals (actual values);
- **T** is a finite set of value types (INT, VARCHAR, etc.)
- *G* is a finite set of graphs;
- *N* is a finite set of nodes;
- *E* is a finite set of edges such that *N* and *E* have no elements in common;
- $\rho : E \rightarrow (N \times N)$ is a total function;
- $\lambda : (N \cup E) \rightarrow \mathrm{SET}(\mathbf{L})$ is a total function;
- $\sigma : (N \cup E) \times \mathbf{K} \rightarrow \mathrm{SET}(\mathbf{V})$ is a partial function;
- $\vartheta : \mathbf{V} \rightarrow \mathbf{T}$ is a function;

Single Graph example

- **L** = {Person, knows}
- **P** = {id, name, since}
- **V** = {“123”, “456”, “Juan Sequeda”, “Marcelo Arenas”, “2010”}
- N = {n1, n2}
- E = {e1}
- ***ρ*** = [*ρ*(*e1*) = (*n1*,*n2*)]
- ***λ*** = [*λ*(*n1*) = *“Person”*, *λ*(*n2*) = *“Person”*, *λ*(*e1*) = *“knows”*]
- ***σ*** = [
  σ(*n1*) = {(*“id”*, *“123”*), (*“name”*, *“Juan Sequeda”*)},
  σ(*n2*) = {(*“id”*, *“456”*), (*“name”*, *“Marcelo Arenas”*)},
  σ(*e1*) = {(*“since”*, *“2010”*)}
  ]

Graphs as Tables

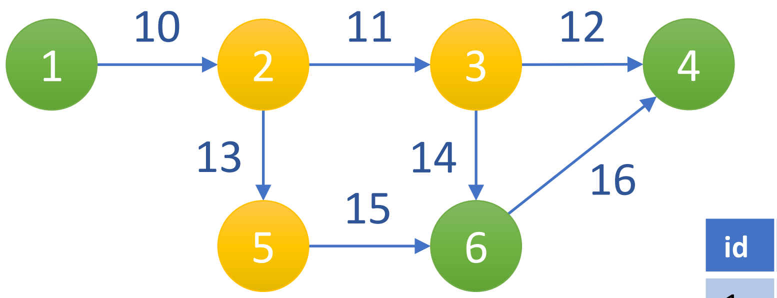
- Complete the picture; integrates different perspectives
- Allows to define semantics based on well-defined relational semantics
- Helps to better see/understand the delta to SQL
- Helps integration/adaption of QL in relational systems
- Suggests one possible implementation

Graphs as Tables

- built-in properties **id$**, **src$**, **dest$**, **graph$**;
- **id$**: ($G$ U N U E) → OID;
- OID(X) = { x.**id$** | $x \in X$ }
- **src$**: E → OID(N);
- **dest$**: E → OID(N);
- **graph$**: N → OID(G).
- two tables **Vertice** and **Edges**:
- the schema of **Vertice** is {**id$**, **graph$**} U {k | $\sigma(n,k)$ is defined; $n \in N$ and $k \in$ **K**}
- the schema of **Edges** is {**src$**, **dest$**} U {k | $\sigma(e,k)$ is defined; $e \in E$ and $k \in$ **K**}
- **Vertice** = $\mathrm{U}_{n \in N} \pi_{\mathrm{S}(\textbf{Vertice})}\{\mathrm{n}\}$ **Edge** = $\mathrm{U}_{n \in N} \pi_{\mathrm{S}(\textbf{Edge})}\{\mathrm{n}\}$

Example

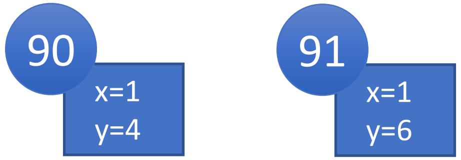
Data graph G



id	label	E.id	E.dest
1	green	10	2
2	yellow	11	3
		13	5
3	yellow	12	4
		14	6
4	green	*no out edges*	
5	yellow	15	6
6	green	16	4

Query

```
SELECT x, y
FROM G (x:green)-[e:+]->(y:green)
```



id	x	y	E.id	E.dest
90	1	4	*no out edges*	
91	1	6	*no out edges*	
⋮				

Dealing with Objects

- Nodes and edges are object with a system managed identity
- Object Immutability
    - For queries, objects (nodes and edges) are immutable
    - Query can create new (transient) objects out of queried data
    - Consequence: Within scope of a query the object identity functionally determines meta type, label, and property values of an object
- Identity Generation
    - Object constructor produces new object identities (OID values)
    - The scope of ID uniqueness is the transaction (query)
    - Repeatability of ID generation is not guarantied

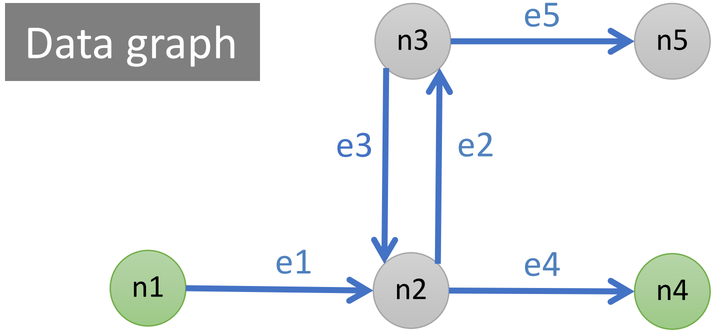
Current Discussion Points

How to represent Paths

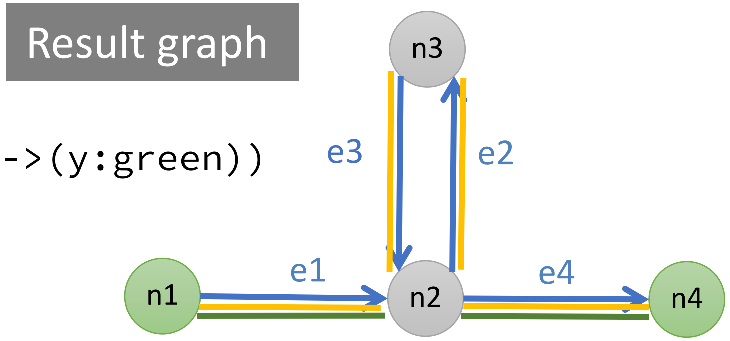
- How to represent path in the data model?
  - Just use existing elements of the data model
    - Keeps data model simple, but complicate interpretation on top
  - As a data type for properties
    - Introduced non-atomic type to properties → complicates language
  - By-elements (actual node and edges) vs. by-reference (list of ids)

- Current favorite: Logical paths
  - Another top level set in the data model
  - Does not contain all paths but just marks paths of users interest

Example: Logical Paths



```
SELECT p
FROM G p=((x:green)-[e:*]->(y:green))
```



- $G = (N, E, P, \rho, \lambda, \delta, \sigma)$
- N = {n1,n2,n3,n4,n5}
- E = {e1,e2,e3,e4,e5}
- $P = \varnothing$
- ρ(e1)=(n1,n2), ρ(e2)=(n2,n3), ρ(e3)=(n3,n2), ρ(e4)=(n2,n4), ρ(e5)=(n3,n5)

- $G' = (N', E', P', \rho', \lambda', \delta', \sigma')$
- N' = {n1,n2,n3,n4}
- E' = {e1,e2,e3,e4}
- P' = {p1,p2,…}
- ρ'(e1)=(n1,n2), ρ'(e2)=(n2,n3), ρ'(e3)=(n3,n2), ρ'(e4)=(n2,n4)
- **δ'(p1) = [e1,e4]**
  δ'(p2) = [e1,e2,e3,e4]

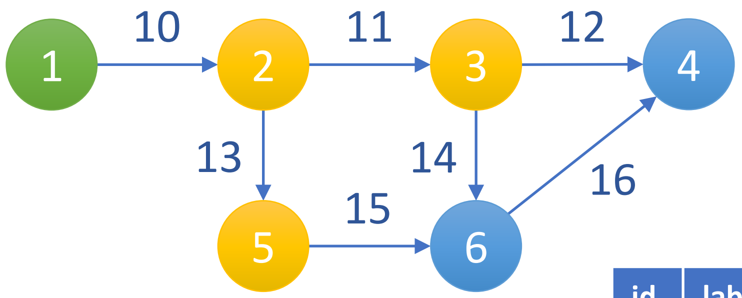
Projection

- Relational-like projection
  - Limit result of single query to single type of nodes and out edge

  -
    ```
    SELECT X.name, X.gender | length(p) AS sim
    FROM G (X:Person)-[p:friend+]->(Y:Person)
    GRAPH BY Y
    ```
  - UNION allows to assemble more complex graphs
  - Unclear how edge targets are projected if of another node type
- Graph transformation-like projection (graph projection for short)
  - Allows to project to multiple node and edge types in one go
  - Result specified by a subgraph pattern
  - Intuitive, very graphy, not 100% orthogonal to UNION

Example: Graph Projection

Data graph G

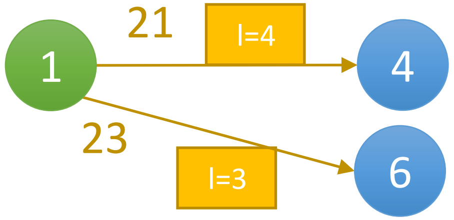


id	label	E.id	E.dest
1	green	10	2
2	yellow	11	3
		13	5
3	yellow	12	4
		14	6
4	blue	*no out edges*	
5	yellow	15	6
6	blue	16	4

Query:

Creates new edge in result graph

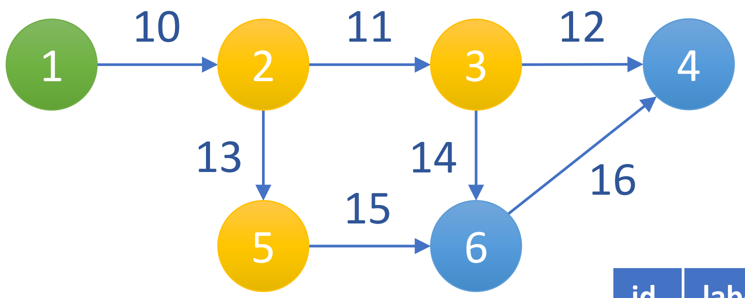
```
SELECT (x)-[:{l=length(p)}]->(y)
FROM G (x:gree)-[CHEAPEST p:*]->(y:blue)
```



id	label	E.id	E.dest	E.l
1	green	21	4	4
		23	6	3
6	blue	*no out edges*		
4	blue	*no out edges*		

Example: Graph Projection w/ Aggregation

Data graph G



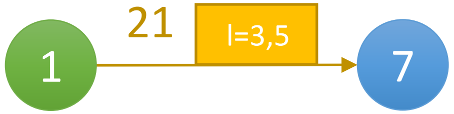
id	label	E.id	E.dest
1	green	10	2
2	yellow	11	3
		13	5
3	yellow	12	4
		14	6
4	blue	*no out edges*	
5	yellow	15	6
6	blue	16	4

Query:

Creates one new edge per x group

Creates one new node per x group

```
SELECT (x)-[:{l=AVG(length(p))}]->(z GROUP BY x:blue)
FROM G (x:green)-[CHEAPEST p:*]->(y:blue)
```



id	label	E.id	E.dest	E.l
1	green	21	7	3,5
7	blue	*no out edges*		

Summary

- Accomplished
    - Catalog of functionalities
    - Core data model
    - Principles object identities
- In discussion
    - Data model extension for path representation
    - Principles of graph construction
- Ahead of us
    - Putting pieces together – define semantics of language core
    - Define a syntax
    - Extend core toward advanced concepts (e.g. path with data)