Querying Graphs: A More Natural Data Model

HPTS 2017

alastair.green neo4j.com

Property Graph Databases/Query Engines are spreading

Neo4j 2006

Blueprints/Gremlin Servers (Titan/Janusgraph, Neo4, …)

IBM System G

IBM BIgInsight

OrientDB

Oracle PGX

Datastax DSE Graph 2016

SAP HANA Graph

AgensGraph [PostgreSQL base]

Azure CosmosDB

SQLServer 2017

Redis Graph

Tigergraph

Amazon graph cloud service TBC

Neo4j: A Raft-orchestrated cluster managing graph data

All updates in all replicas are ACID

Tiered read-only replicas

Default Causal Consistency (RYOW)

Can dial down to faster stale reads

Causal Tokens (stateless connections)

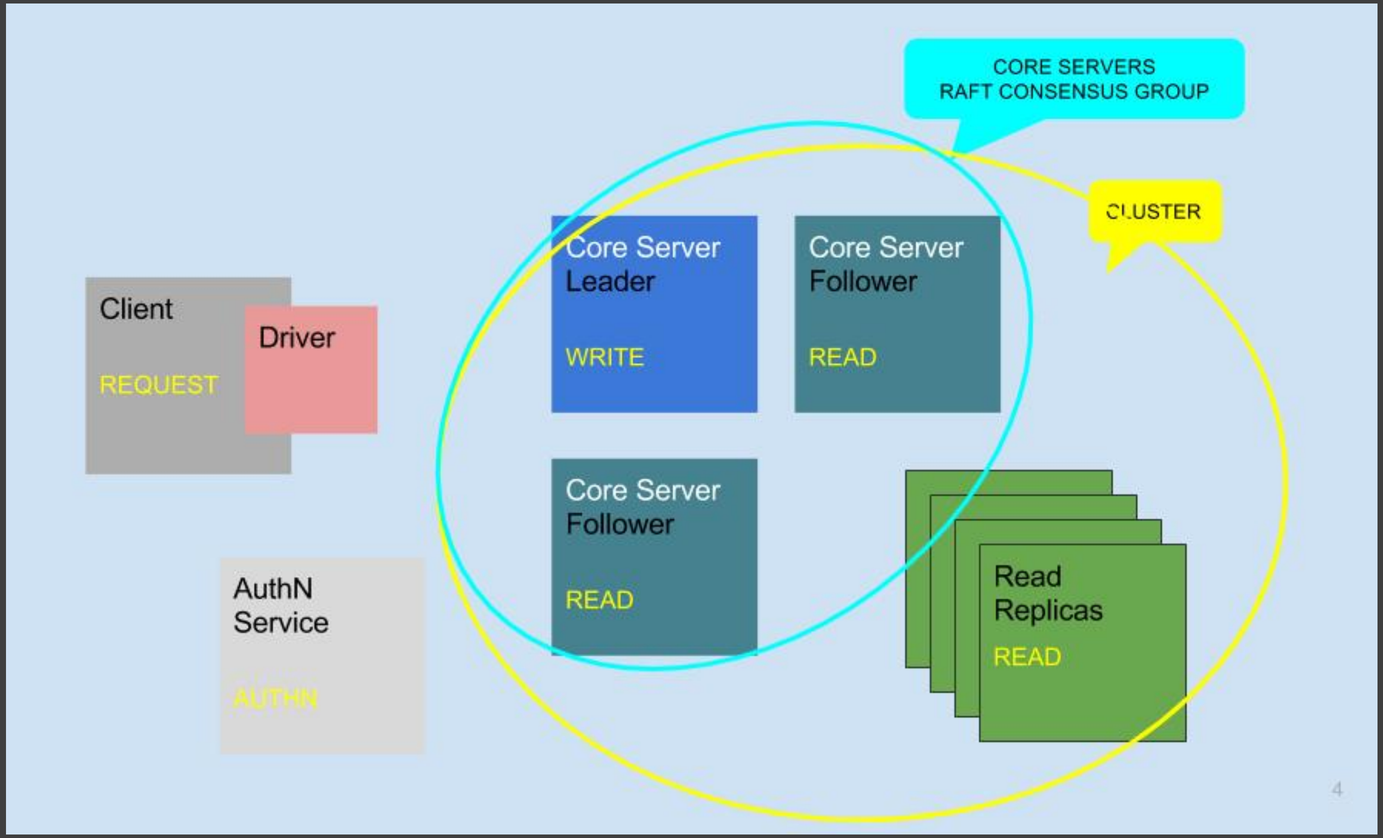
Cluster-managed load-balancing/routing

Hides failures and leader elections from app

Route to read/write endpoints

Route to cluster regions (server groups)

User configured replication paths and routing



Why?

Graph applications like social network or fraud detection or IoT are front of mind

Graph traversals (long chains of joins) are very fast in memory.897

Things have changed since Codasyl

“Big RAM growing faster than Big Data”

~ True for structural graphs (topology) and key properties for predicate evaluation

Property Graphs are good models of reality

Infrastructure networks

Social networks

Business processes

Flow dependencies

Mindmaps

Whiteboards

A superset data model

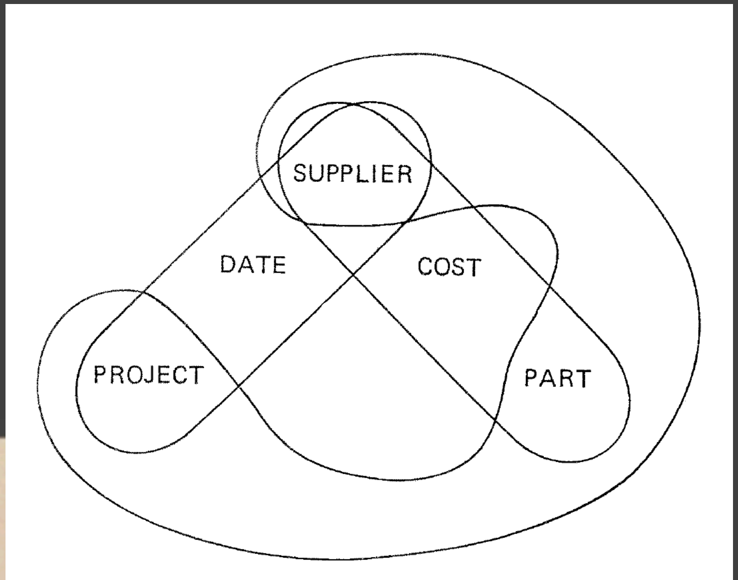
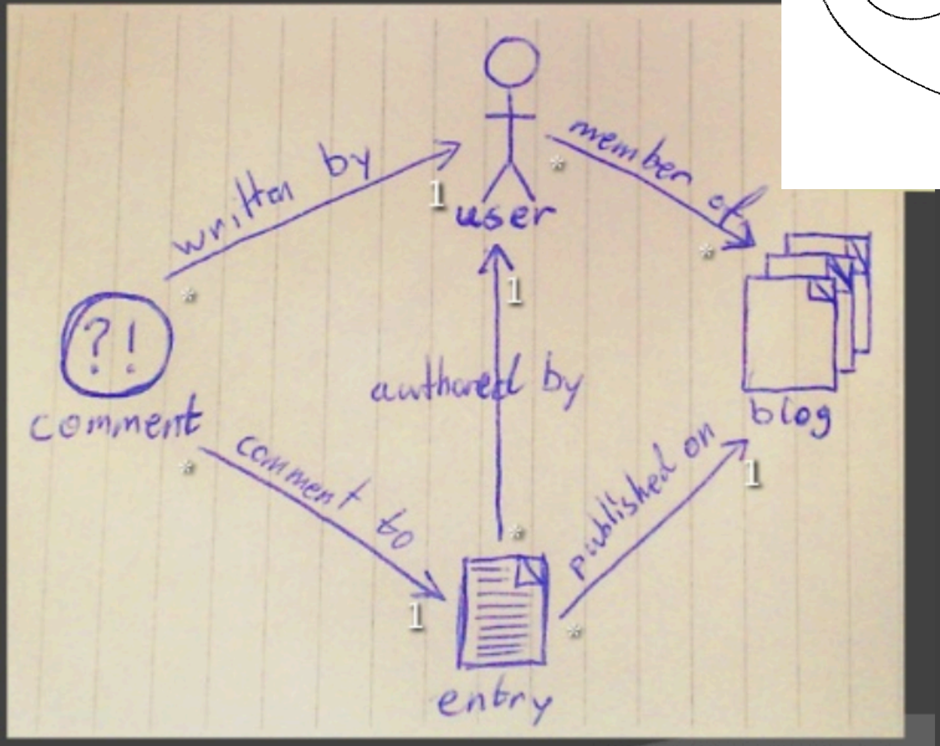


Figure 1,2



The conceptual model and the physical model are very close

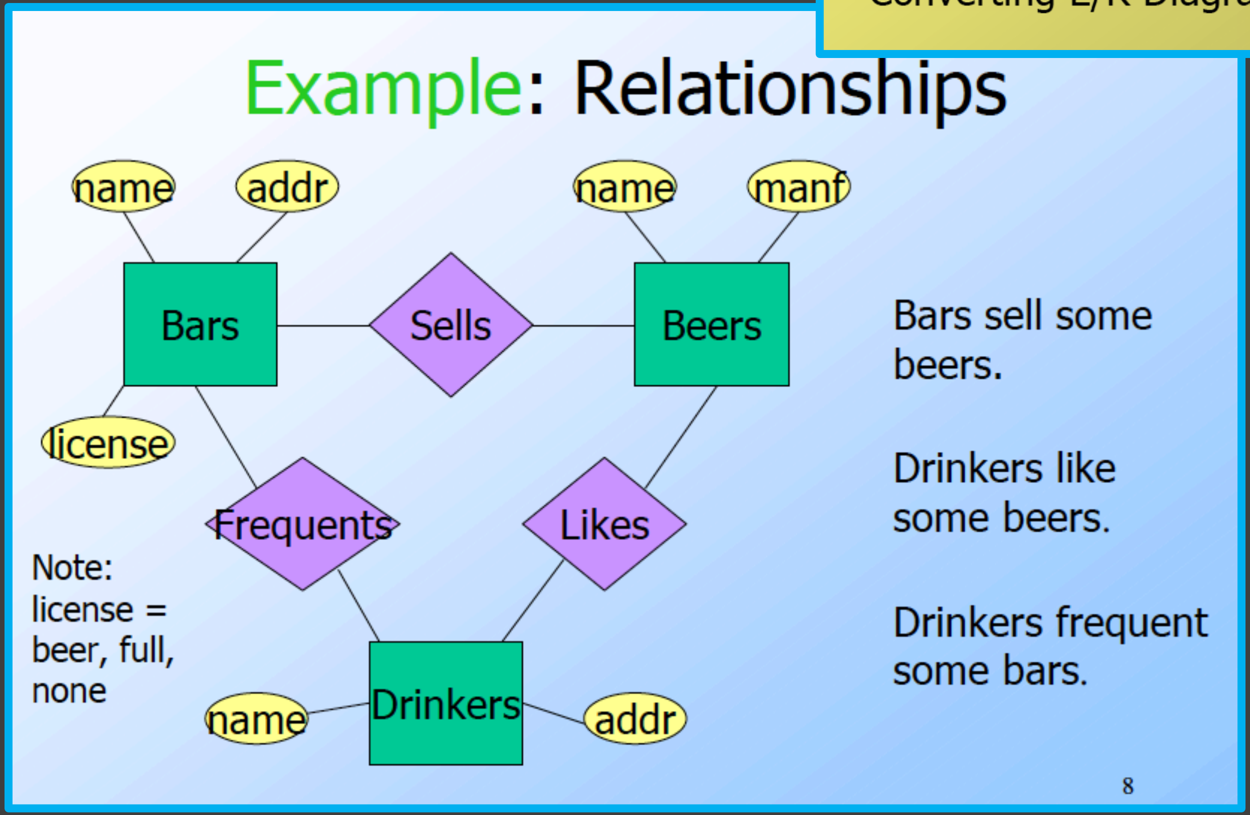
The translation step to a relational schema fades away

Relationships become first-class concerns:

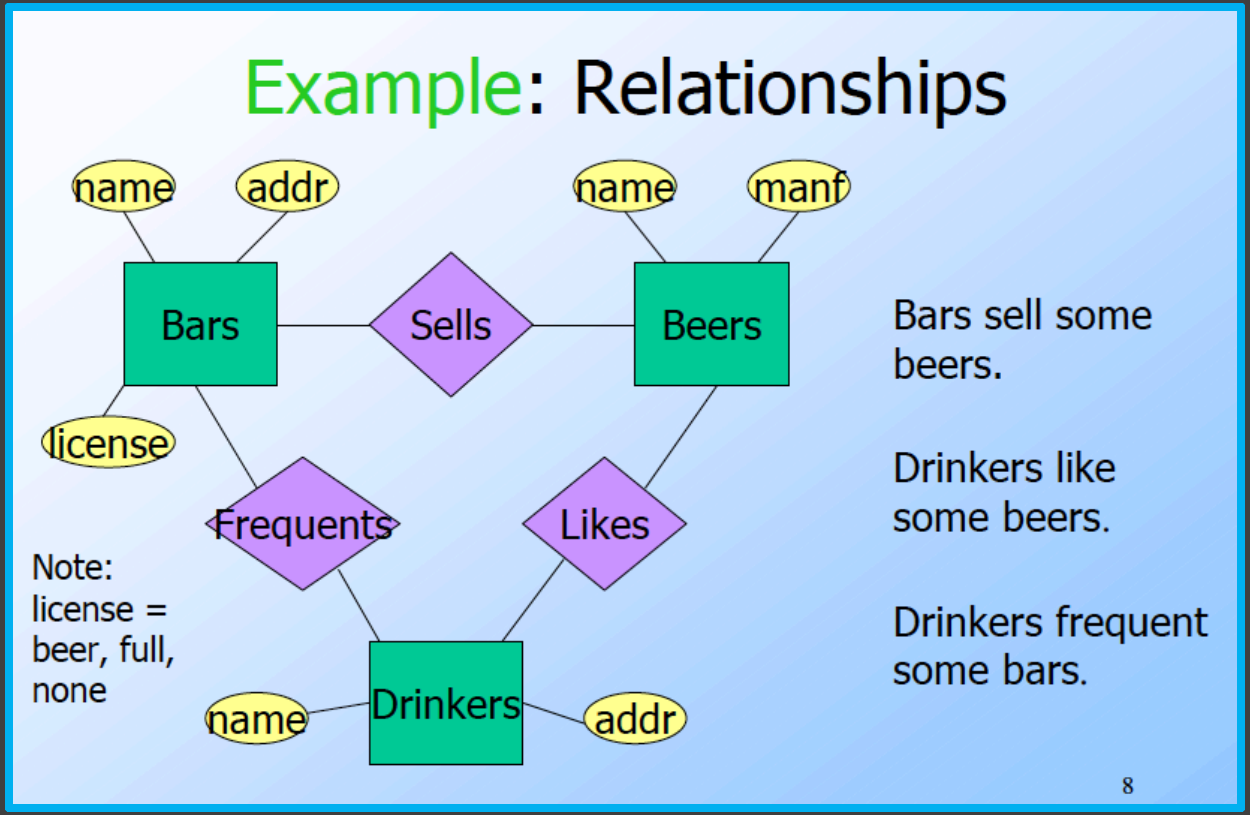
PK-FK and join tables are a disincentive to modelling relationships in the DB

In graph DBs relationship properties are easy

Relationship names can be used in query predicates



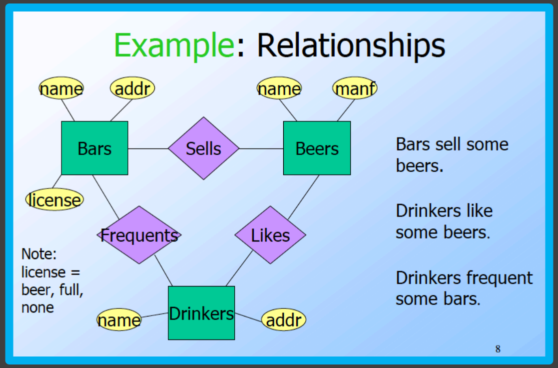
Mapping ER diagram directly to the property graph data model



Mapping ER diagram directly to the property graph data model

Entities Nodes with Labels

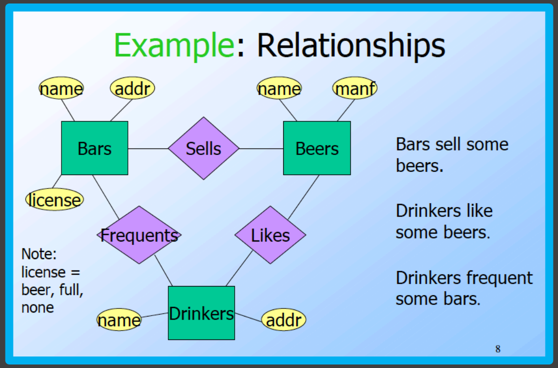
```
// Populate entity set Beers Nodes with label Beers

create (:Beers{name:'Anchor Steam',manf:'Anchor Brewing'})
create (:Beers{name:'Bass IPA',manf:'AB-Interbev'})

// Populate entity set Bars Nodes with label Bars

create (:Bars{name:'The Lamb', addr:"Lamb's Conduit St"})
create (:Bars{name:'Pelican Inn', addr:'Muir Beach'})

match (b:Bars{name:'Pelican Inn'}) set b.license = 'Full’

// Beer-only licences don’t exist in the UK anymore
// No license property on The Lamb: schema optional

// And schema constraints in e.g. Cypher are very limited
```



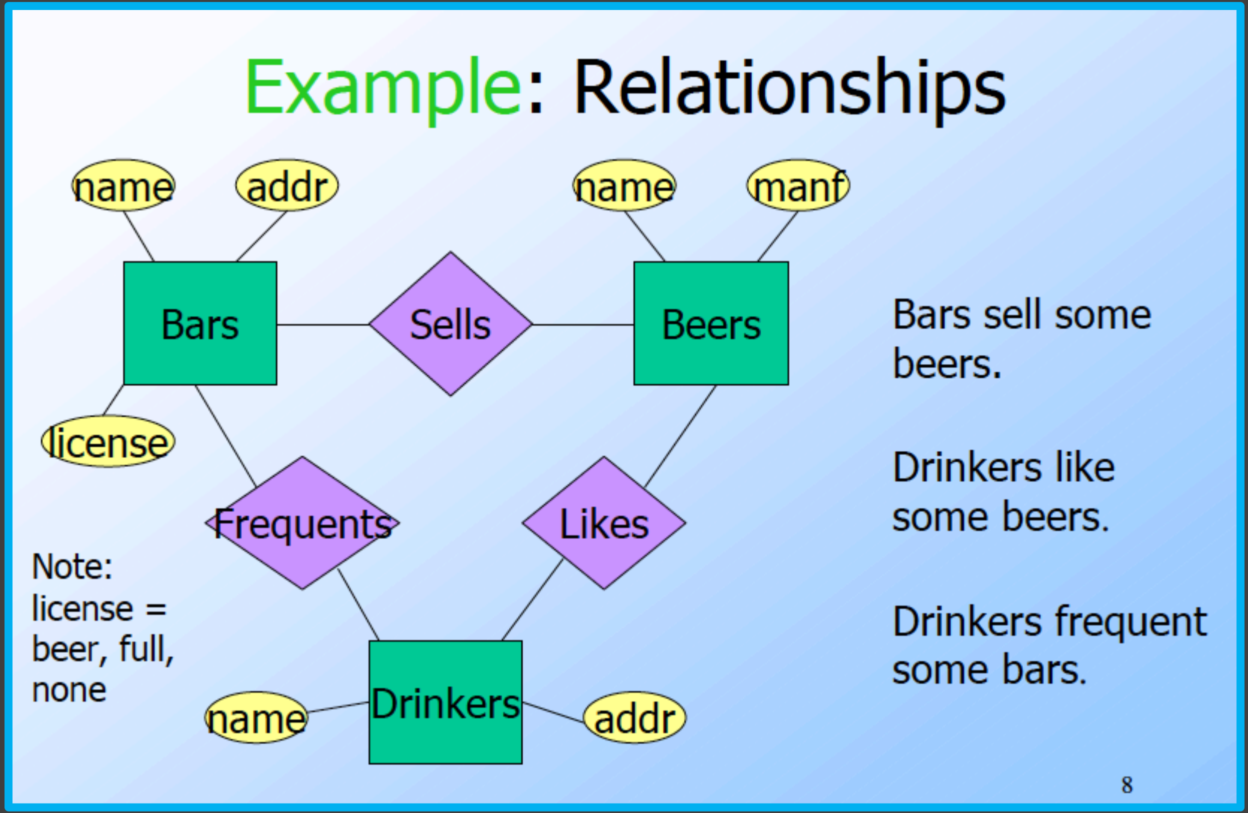
Mapping ER diagram directly to the property graph data model

Relationships With Types

```
// Relationships with type LIKES
match
    (d:Drinkers), (be:Beers)
create
    (d)-[:LIKES]->(be)

// Populate relationship set SELLS with attributes
// Relationships with type SELLS, which has properties

match (b:Bars{name:'Pelican Inn'}),
      (be:Beers{name:'Anchor Steam'})
create
    (b)-
        [:SELLS
             {currency:'USD',
              price:800,
              pourage:'Bottle',
              size:'16oz'}]
    ->(be)

match (d:Drinkers) set d:Ullman // second label
match (b:Bars) set b:Ullman
match (be:Beers) set be:Ullman
```

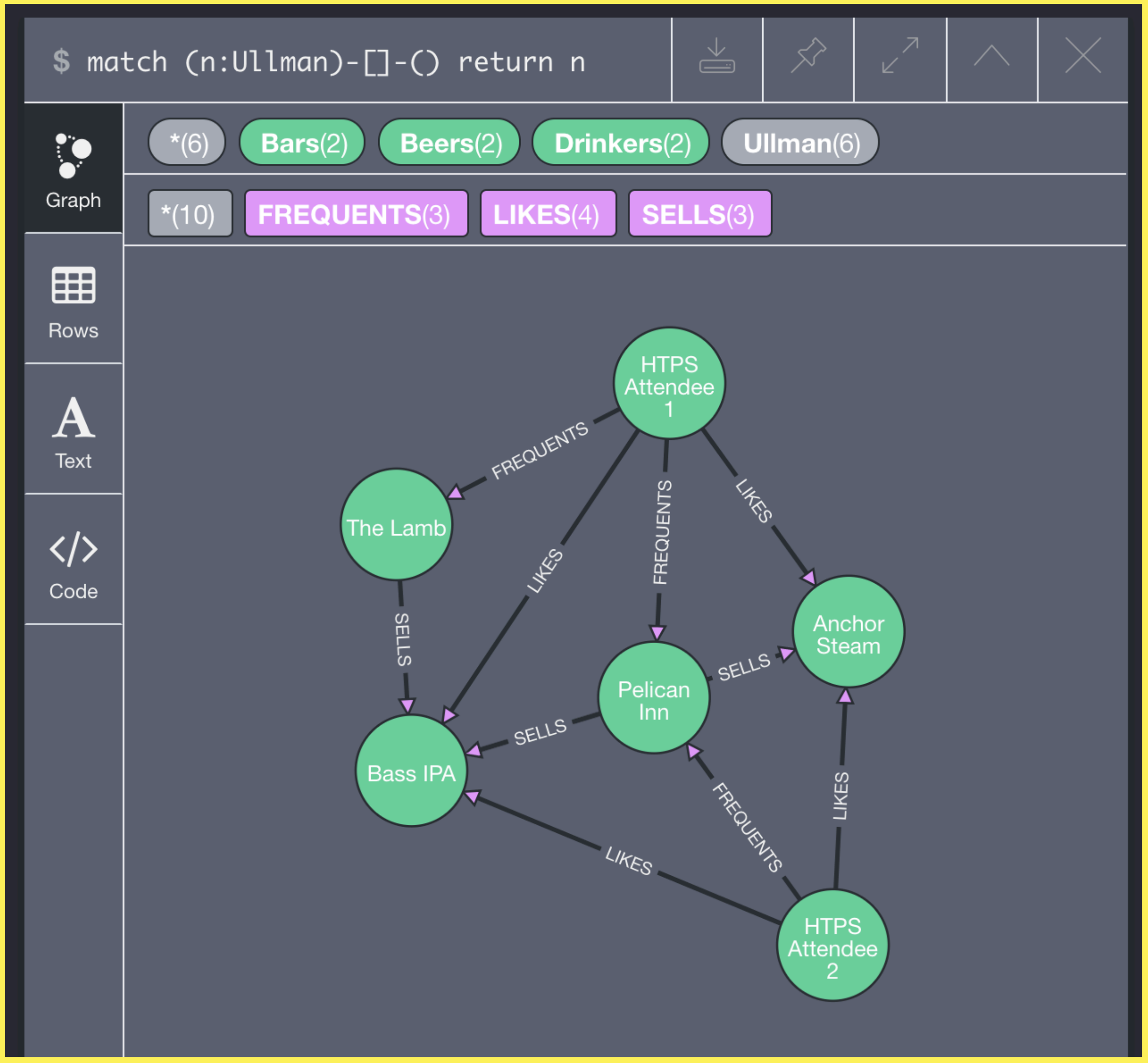


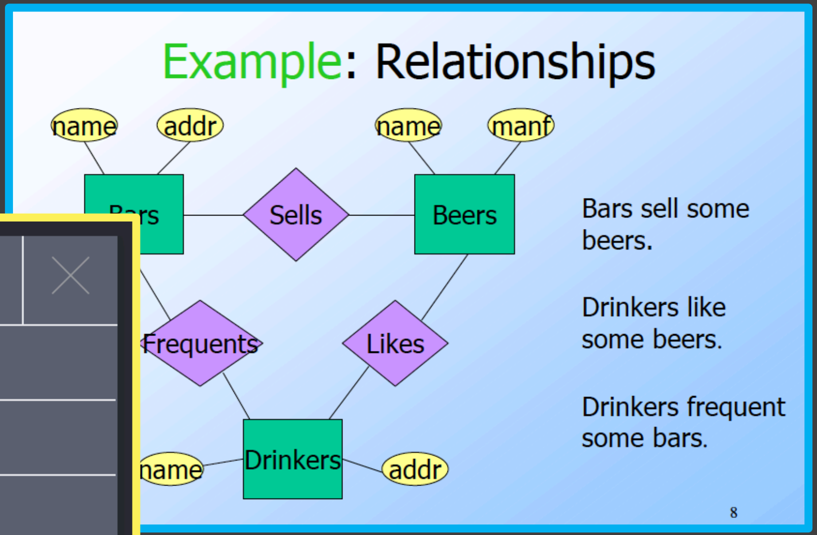
Mapping ER diagram directly to the property graph data model



Ullman Drinking Model

Nodes and Edges



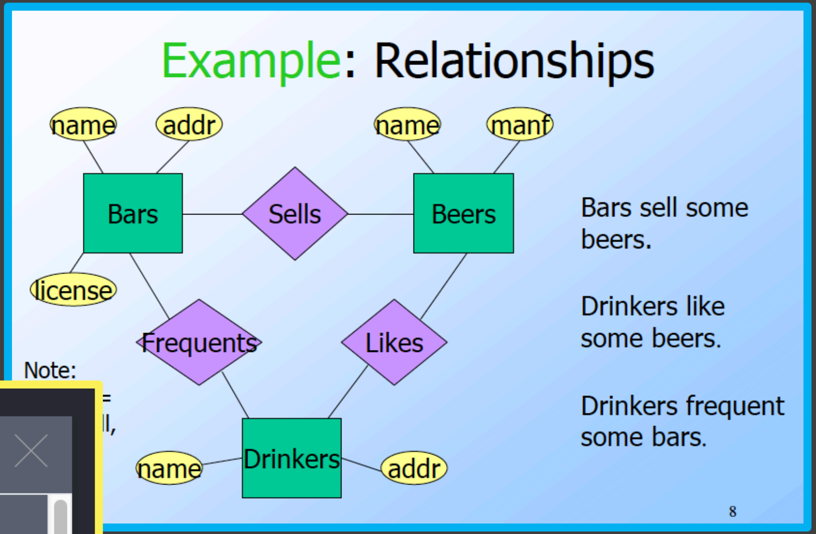


And their attributes or graph “entity properties”

```
$ match (s:Ullman)-[r]-(e) return s, r, e
```

Graph | Rows | Text | Code

s	r	e
name Bass IPA manf AB-Interbev	ranking 0.9	name HTPS Attendee 1 addr London
name Bass IPA manf AB-Interbev	ranking 0.6	name HTPS Attendee 2 addr Bay Area
name Bass IPA manf AB-Interbev	size 20oz price 500 currency GBP pourage Draught	license Full name The Lamb addr Lamb's Conduit St



Precursors and precedents: standards

SPARQL 1.1 Query Language

W3C Recommendation 21 March 2013

Abstract

RDF is a directed, labeled graph data format for representing information in the Web. the SPARQL query language for RDF. SPARQL can be used to express queries acro natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for with their conjunctions and disjunctions. SPARQL also supports aggregation, subque extensible value testing, and constraining queries by source RDF graph. The results

Eric Prud'hommeaux, W3C

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Copyright © 2013 W3C® (MIT, ERCIM, Keio, Beihang), All Rights Reserved. W3C liability, trademark and document use ru

complish a parts
ng the "find next,
owner in the other
der should be able

matter where the part is used. The structure can also be regarded as an acyclic directed graph, as shown in Figure 14, where the

March 1976

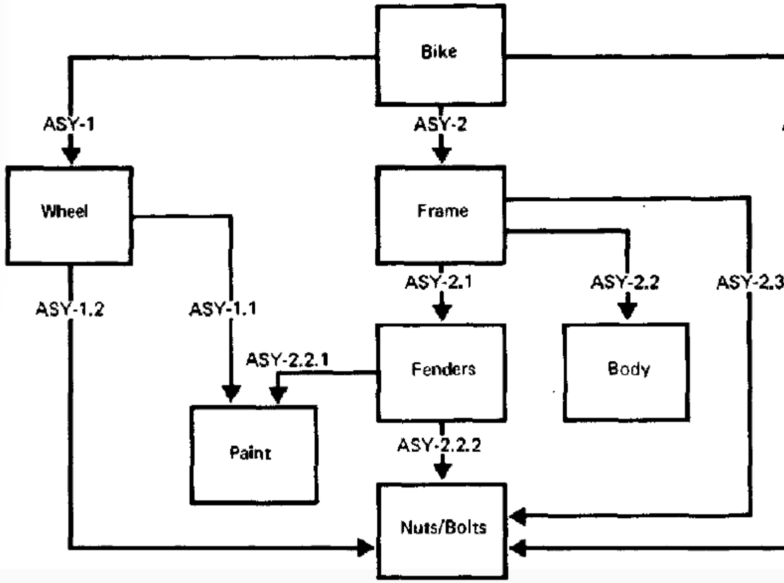


FIGURE 14. Parts explosion as a graph.

Industrial property graph query languages

Neo4j Cypher

Gremlin (Titan, Janusgraph)

IBM System G Gremlin

OrientDB SQL proprietary extensions, OO

Oracle PGX PGQL, Gremlin

Datastax DSE Graph Gremlin

SAP HANA Graph Cypher

AgensGraph [PostgreSQL base] Cypher, SQL hybrid

Azure CosmosDB Gremlin

SQLServer 2017 SQL proprietary extensions

Redis Graph Cypher

Tigergraph SQL proprietary extensions

Amazon graph cloud service TBC

Cypher openCypher

Gremlin Apache Tinkerpop

PGQL Oracle

SQL 2020

What about

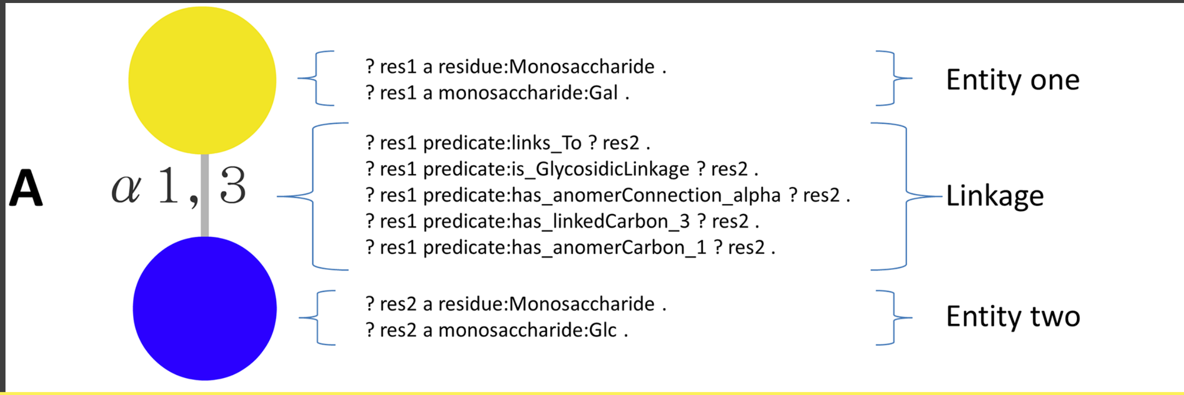
SPARQL

Gremlin?

Property graphs easier to work with than RDF triples

But there is much to learn from the RDF/SPARQL world

Property Graph vs RDF Triple Store: A Comparison on Glycan Substructure Search
Davide Alocci, et al. PLOS ONE, 2015



MATCH

P = (Component1:MONOSACCHARIDE)-[Linkage:GLYCOSIDIC]->(Component2:MONOSACCHARIDE)

WHERE

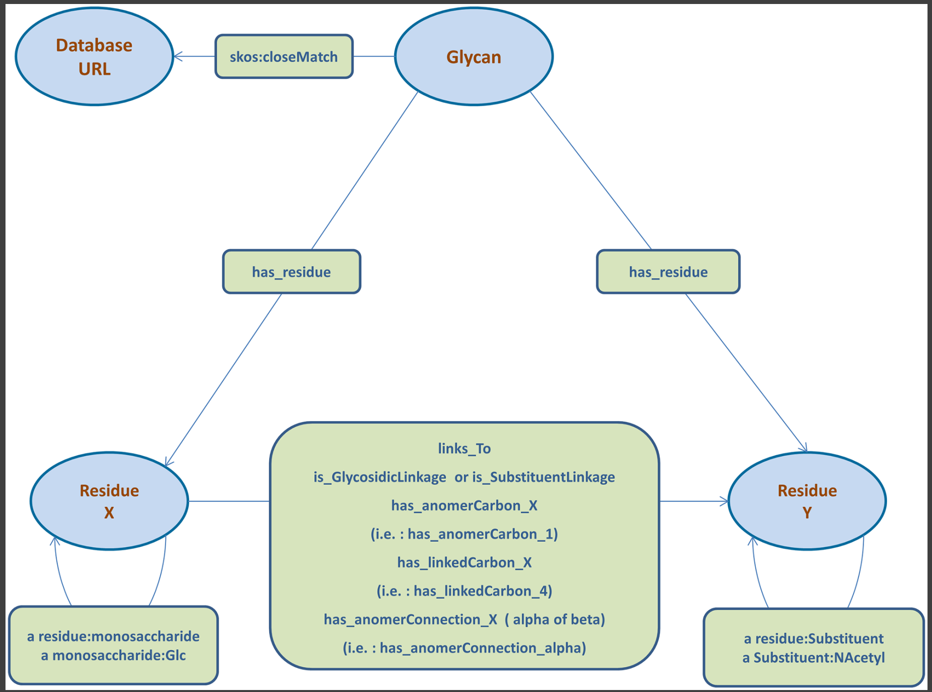
Linkage:anomericity = alpha' and
Linkage:anomericCarbon = 1 and
Linkage:.linkedCarbon = 3 and

Linkage

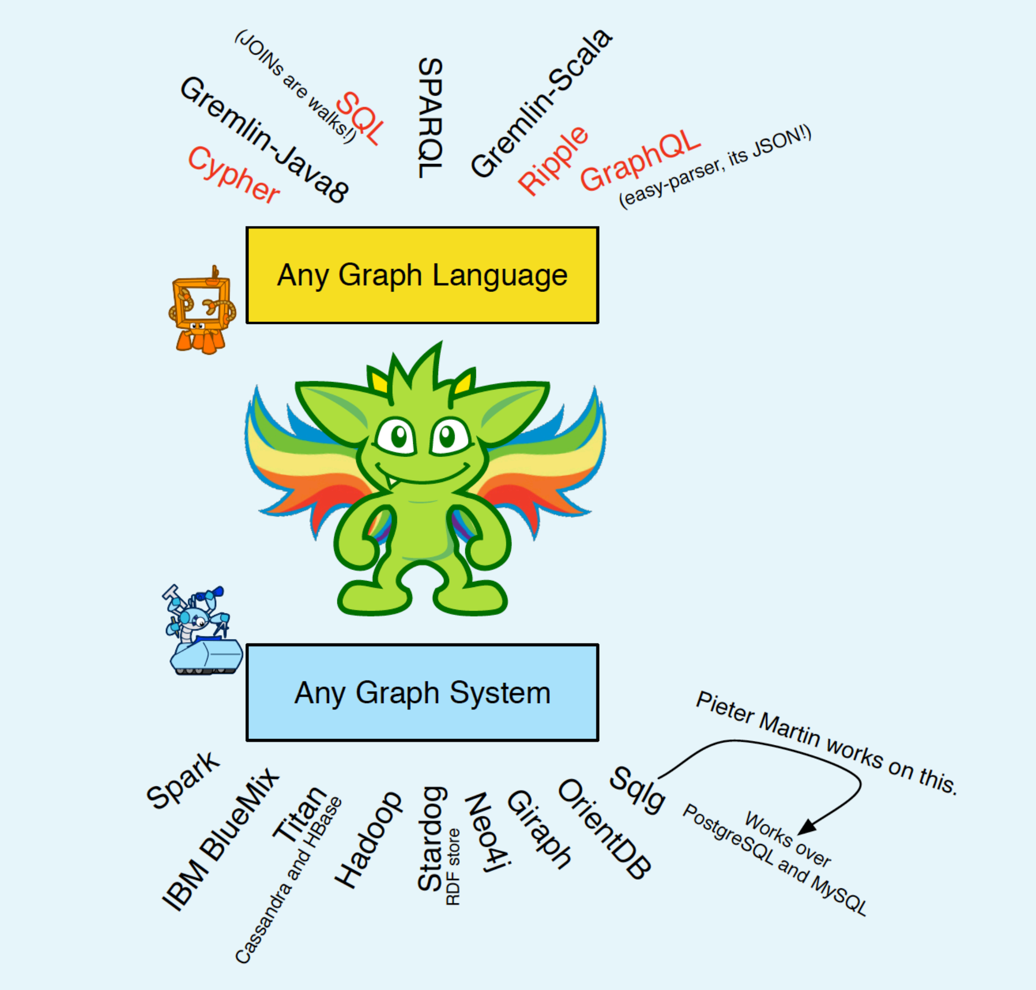
Component1.name = 'Gal'
and Component2.name = 'Glc‘

Entities

Graph Pattern



Gremlin’s Architecture + API



Gremlin is a “traversal API and a traversal engine”

You can implement your own engine (CosmosDB)

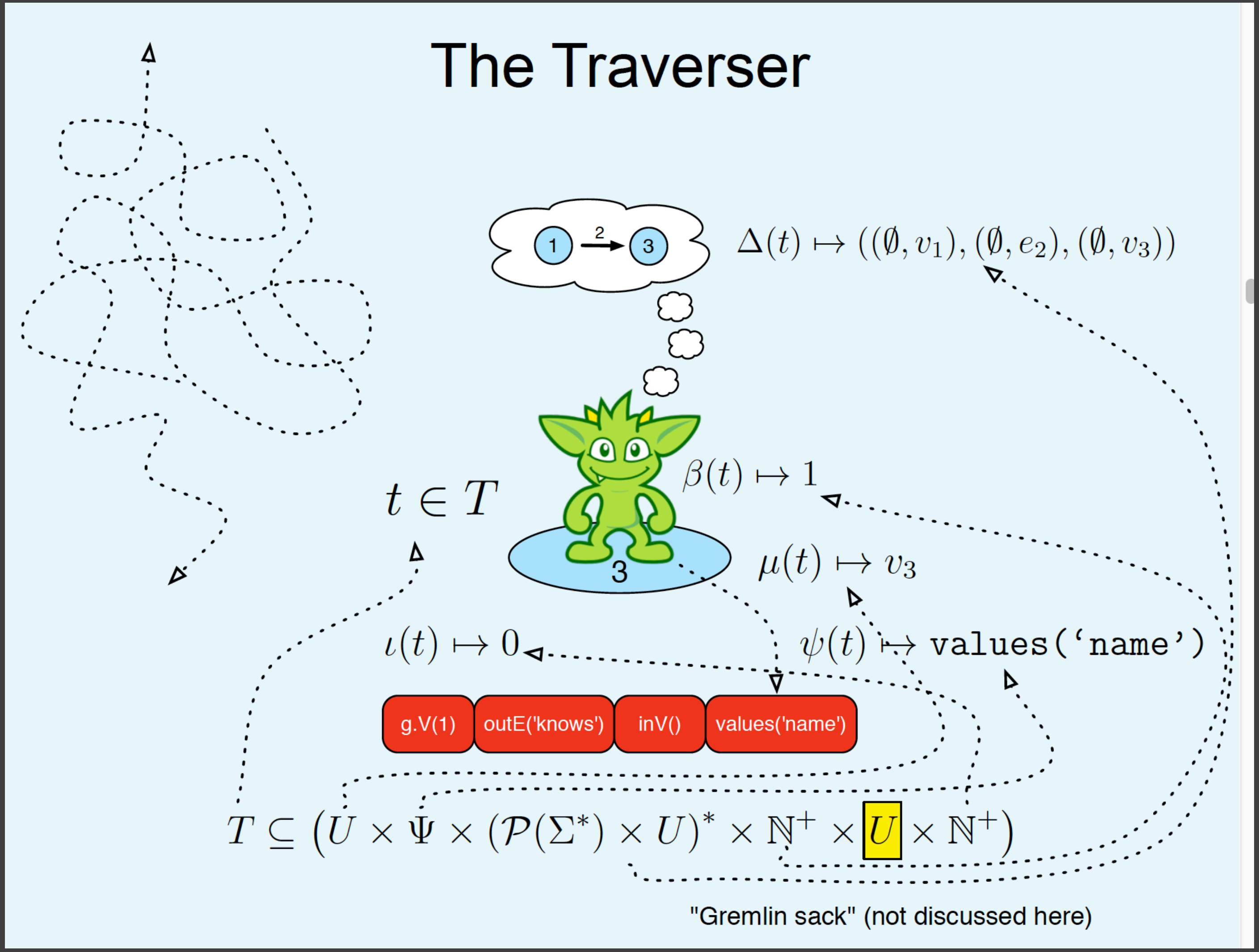
As exposed in a GPL, Gremlin feels like a functional, “fluent”, imperative API

```
MATCH (ss1:StarSystem {name: 'Solar System'})
MATCH (ss1)-[r:DISTANCE_TO]-(ss2:StarSystem)
WHERE r.ly <= 5
RETURN ss2.name;


g.V()
    .as('ss1').hasLabel('StarSystem')
        .where(__.as('ss1').values('name').is(eq('Solar System')))
    .as('ss1')
        .bothE('DISTANCE_TO').as('r')
        .otherV().as('ss2').hasLabel('StarSystem')
    .where(__.as('r').values('ly').is(lte(5)))
    .select('ss2');
```

It does not attempt to fill the ecological niche of SQL

You can implement your own high-level language which compiles to Gremlin’s API (e.g. SPARQL, Cypher)

Gremlin: embedded DSL, not "SQL for Graphs"



Gremlin is not very approachable for database developers used to SQL

The emphasis of Gremlin is more on functionality and less on surface

Gremlin is an impressive creation, but lacks ease-of-use for the wider market

Property Graph Data Model Query Languages are a hot topic

Neo4j Cypher

Gremlin (Titan, Janusgraph)

IBM System G Gremlin

OrientDB SQL proprietary extensions, OO

Oracle PGX PGQL, Gremlin

Datastax DSE Graph Gremlin

SAP HANA Graph Cypher

AgensGraph [PostgreSQL base] Cypher, SQL hybrid

Azure CosmosDB Gremlin

SQLServer 2017 SQL proprietary extensions

Redis Graph Cypher

Tigergraph SQL proprietary extensions

Amazon graph cloud service TBC

Cypher openCypher

Gremlin Apache Tinkerpop

PGQL Oracle

GCORE LDBC QL Task Force

SQL 2020

Some Key Issues

Compositional Queries

Sub-Queries/Views

Multiple Graphs

Path Regular Expressions

Virtual Paths

"Morphism" + tractability

Schema constraints

SQL

Industry standards initiatives

Cypher openCypher

SQL 2020

incits Where IT all begins InterNational Committee for Information Technology Standards

Groups ProjectView Committee Participation Log Out

Workspace » All Groups » My Groups » DM32 Data Management and Interchange » DM32.2 - Database » Ad Hoc on SQL Extensions Property Graphs »

Document Details - sql-pg-2017-0047r1-SQL-Graph-Query-Procedures.pdf

Group Home | Roster | Calendar | Documents | Action Items | Email

RESOLUTION 03-001- Changes to Projects

SC 32 requests its Secretariat to seek authorization from JTC 1 to implement the follow existing standards projects:

a) Establishment of Project Split*

Project #	Title	Reason
9075-16	SQL Property Graph Queries (SQL/PGQ)	See Justification

***Include the scope of the original project**

Part 16 is consistent with the original scope of the ISO 9075 project.

Justification: WG3 is investigating adding support to 9075 SQL for querying, storing, and modifying data in Property Graph format. This work is developed in conjunction with vendors in this space and the Linked Data Benchmark Council (LDBC) Graph QL Task Force. WG3 expects to have a Working

Third openCypher Implementers Meeting - 13 November 2017

The third face-to-face meeting for people, projects and organizations interested in participating in the ope goal of creating a standard language based on Cypher for querying graphs.

openCypher Meetup - 25 October 2017

We will present the major new features which will soon become part of Cypher.

oC *Proposal*

Second openCypher Implementers Meeting - 10 May 2017

The second face-to-face meeting for people, projects and organizations interested in participating in the openCypher project, with the goal of creating a standard language based on Cypher for querying graphs.

First openCypher Implementers Meeting - 8 February 2017

Overlapping participation in these two initiatives and the research efforts of LDBC QL Task Force

Research roots and activity

Survey of Graph Database Models

RENZO ANGLES and CLAUDIO GUTIERREZ

Universidad de Chile

Graph database models can be defined as those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors. These models took off in the eighties and early nineties alongside object-oriented models. Their influence gradually died out with the emergence of other database models, in particular geographical, spatial, semistructured, and XML. Recently, the need to manage information with graph-like nature has reestablished the relevance of this area. The main objective of this survey is to present the work that has been conducted in the area of graph database modeling, concentrating on data structures, query

Querying Graphs with Data

LEONID LIBKIN, University of Edinburgh
WIM MARTENS, Universität Bayreuth
DOMAGOJ VRGOČ, PUC Chile and Center for Semantic Web

We use several versions of XPath-like languages for graph databases, all of them collectively named GXPath. Like XPath (or closely related logics such as PDL and CTL*), all versions of GXPath have node tests and path formulae, and as the basic navigational axes they use letters from the alphabet labeling graph edges. On top

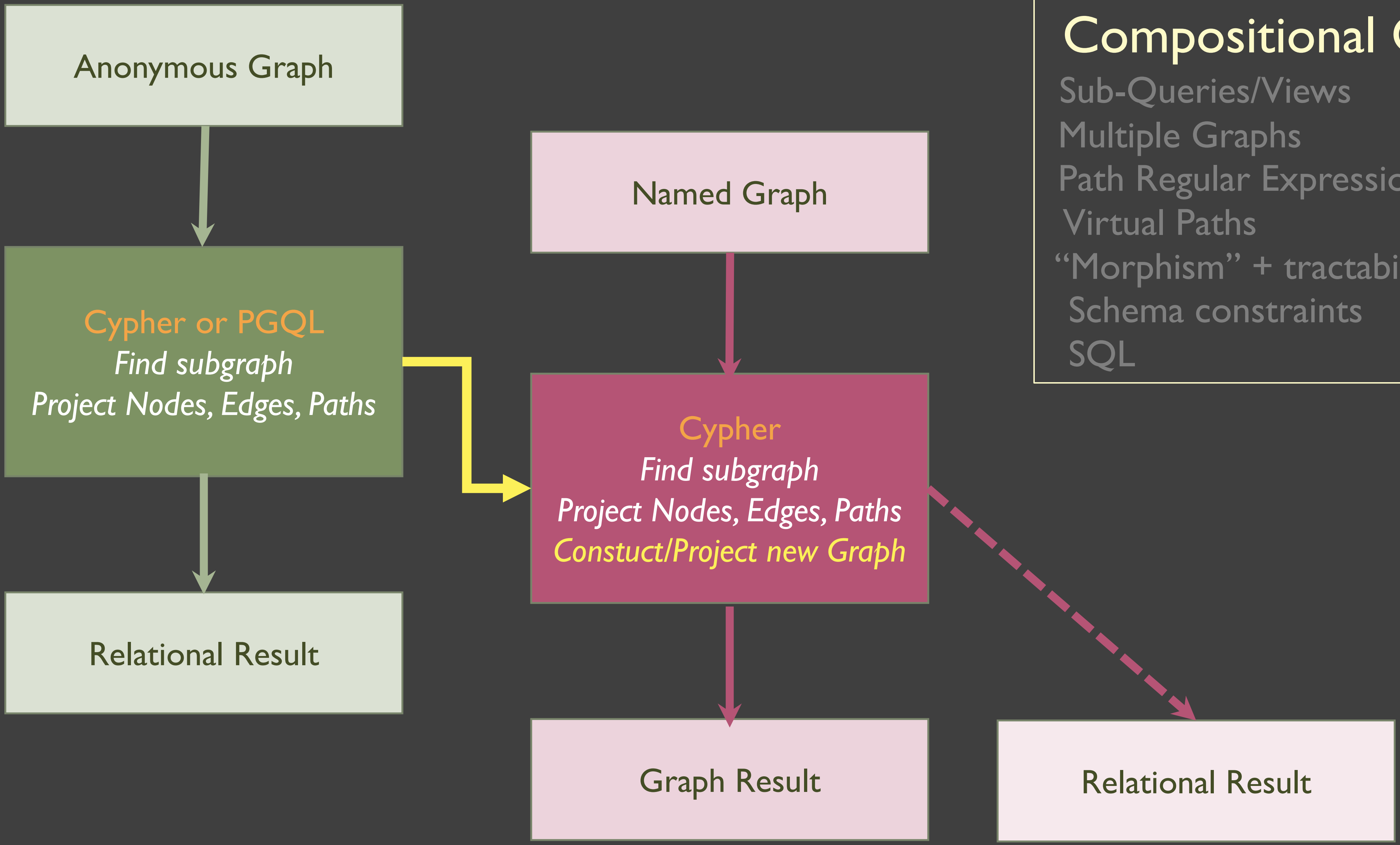
two categories. One views graphs as a particular kind of relational data and uses traditional relational mechanisms for querying. The other concentrates on querying the topology of the graph. These approaches, however, lack the ability to *combine* data and topology, which would allow queries asking how data changes along paths and patterns enveloping it.

Query Languages for Graph Databases

Peter T. Wood
Department of Computer Science and Information Systems, Birkbeck, University of London
ptw@dcs.bbk.ac.uk

ABSTRACT

Query languages for graph databases started to be investigated some 25 years ago. With much current data, such as linked data on the Web and social network data, being graph-structured, there has been a recent resurgence in interest in graph query languages. We provide a brief survey of many of the graph query languages that have been proposed, focussing on the core functionality provided in these languages. We also consider issues such as expressive power and the computational complexity of query evaluation.



Compositional Queries

Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
"Morphism" + tractability
Schema constraints
SQL

Compositional Queries

Sub-Queries/Views

Multiple Graphs

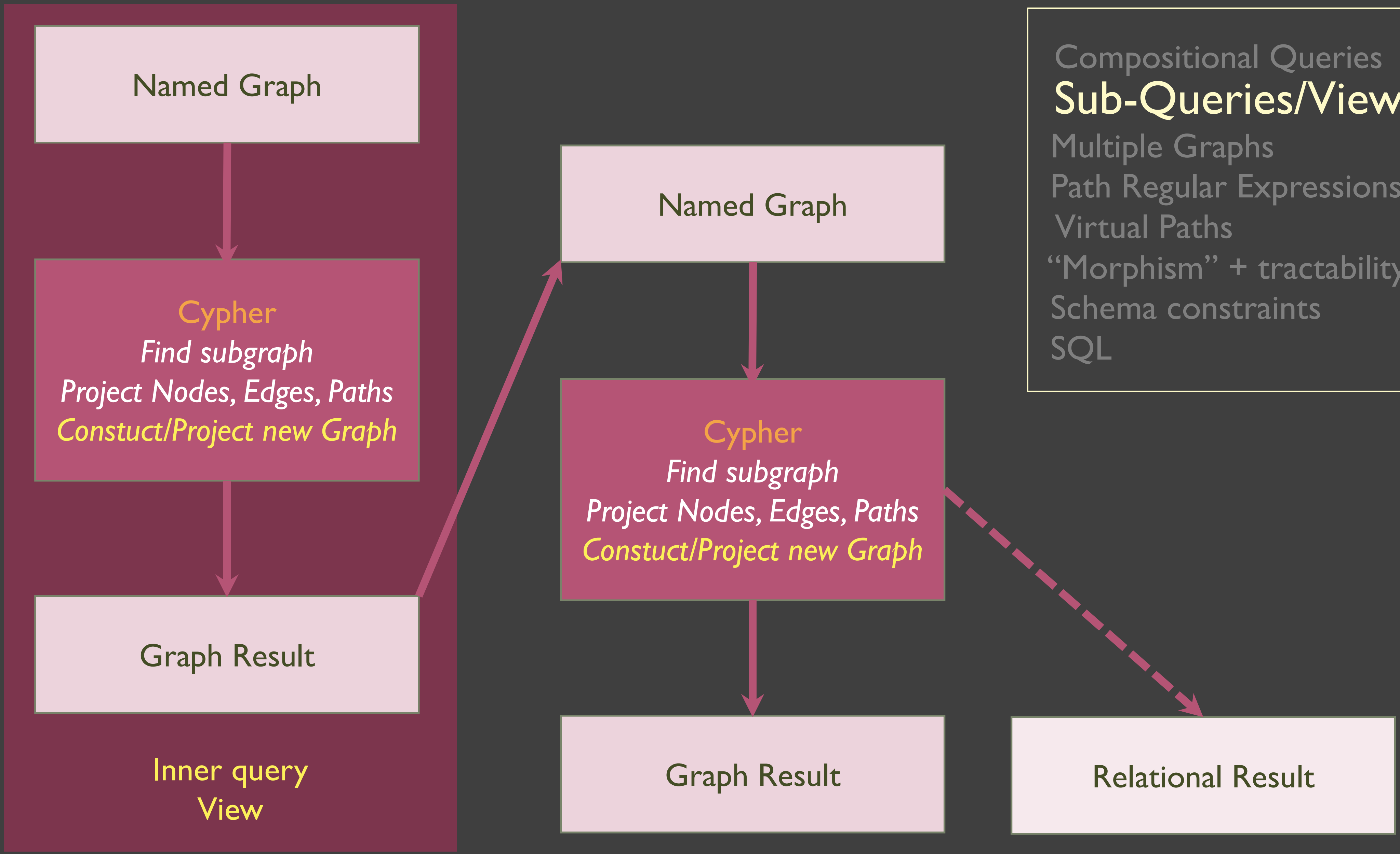
Path Regular Expressions

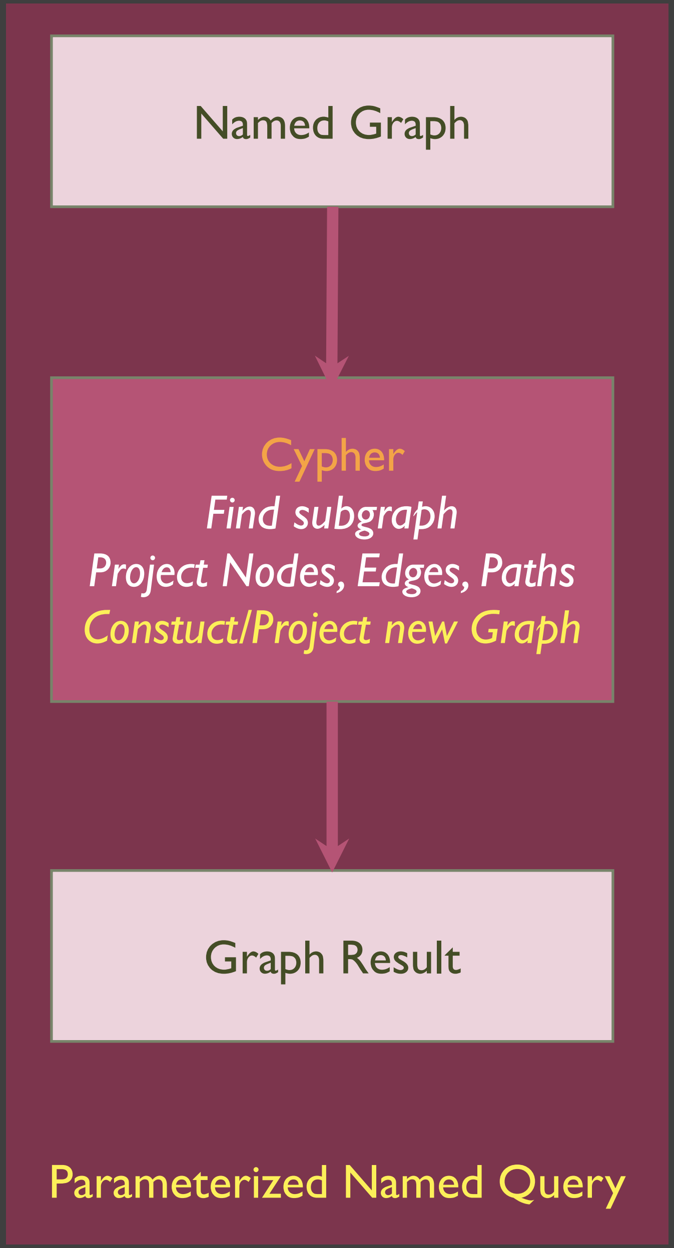
Virtual Paths

“Morphism” + tractability

Schema constraints

SQL





Compositional Queries
Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
"Morphism" + tractability
Schema constraints
SQL

An idea in openCypher

Named query

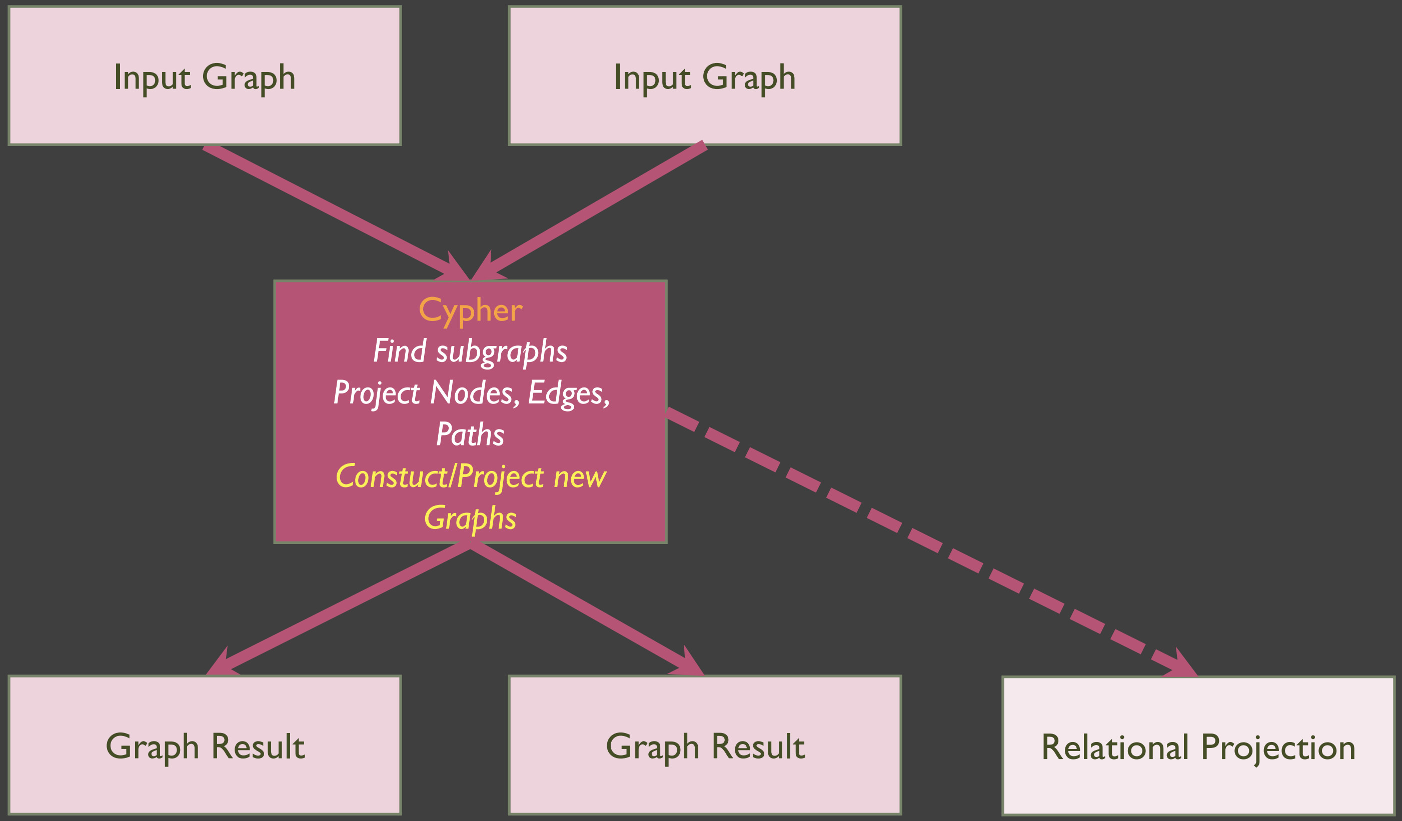
With known parameter values

Can have properties, and edge-order ➔ its value is a graph

Use as HOF for query statements that require a graph input

Can represent any sub-graph/projected graph including paths

Use case: Line in a Subway/Metro system, which has a budget



Compositional Queries
Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
"Morphism" + tractability
Schema constraints
SQL

Set operations

Merging graphs

Immutable / Mutable

Deep / Shallow Projection

Access Control

Database "instances"

graph:// URIs

Predicates on Edge Label `()-/:FOO/-()`

Predicates on Nodes `()-/(:Alpha{beta:'gamma'})/-()`

Alternatives `()-/:FOO | :BAR | :BAZ/-()`

Sequence `()-/:FOO :BAR :BAZ/-()`

Grouping `()-/:FOO | [:BAR :BAZ]/-()`

Direction `()-/<:FOO :BAR <:BAZ>/->()`

Any Edge `()-/-/-()`

Repetition

`()-/:FOO? :BAR+ :BAZ* :FOO*3.. :BAR*1..5/-()`

Predicates on Edge Properties

`()-/ [- {some:'value'}] /-()`

(applies to all edges matched by the group,for more complex predicates use Named Path Patterns)

Negation `()-/[^:FOO :BAR]/-()`

Compositional Queries
Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
"Morphism" + tractability
Schema constraints
SQL

http://www.opencypher.org/ocig2

tobias@neo4j.com

Named Path Patterns

```
PATH PATTERN not_a_hater = (x)
WHERE NOT EXISTS { (x)-[:HATES]->() }

MATCH (you)
         -/:KNOWS ~not_a_hater :KNOWS/-
      (friend_of_a_friendly_friend)
```

Can be used in the definition of “higher” path patterns

Compositional Queries
Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
“Morphism” + tractability
Schema constraints
SQL

[PGQL]()

Oskar van Rest, Oracle

http://www.opencypher.org/ocig2

tobias@neo4j.com

Descriptions of an output from pattern matching

Walk Repeated nodes, Repeated edges

Trail Repeated nodes, No repeated edges

Path No repeated nodes, No repeated edges

The matching "morphisms"

Homomorphism Repeated nodes, Repeated edges

Edge-Isomorphism Repeated nodes, No repeated edges

Node-Isomorphism No repeated nodes, No repeated edges

Edge-isomorphism is also known as Cyphermorphism

The more you get back, the cheaper the computation

Projected graph construction requires deduplication …

One of many current discussions about allowing generally intractable queries to be formulated: e.g. GCORE from LDBC

Compositional Queries
Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
"Morphism" + tractability
Schema constraints
SQL

http://www.opencypher.org/ocig3

stefan.plantikow@neo4j.com

Graph query languages are historically indifferent to schema

SPARQL is schemaless

Cypher is schema optional

Some schema in Neo4j Cypher for indexes and node keys

Plus required properties for a label

Types of properties are not enforceable (yet)

Compositional Queries
Sub-Queries/Views
Multiple Graphs
Path Regular Expressions
Virtual Paths
"Morphism" + tractability
Schema constraints
SQL

Interesting constraints, not just for designers but for query optimization

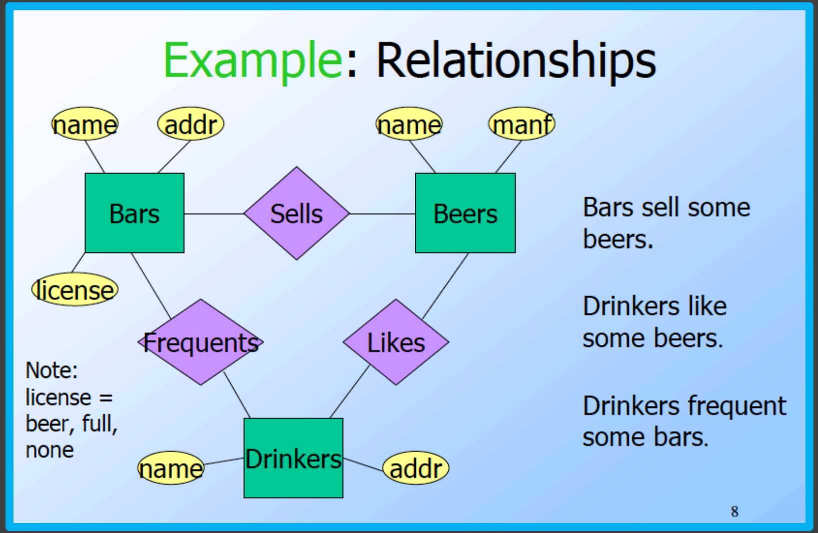
Name, type, optionality of properties by label

Node-edge-node combinations by label

Label combination or inheritance

Cardinality of relationships

Back to the Entity-Relationship Diagram …



Four broad areas under discussion in INCITS SQL Ad-Hoc

”Foreign Languages” like Cypher (or SPARQL or XQuery)

How to define, store and refer to a graph(s) in SQL

Pattern matching and whether to embed SQL predicates

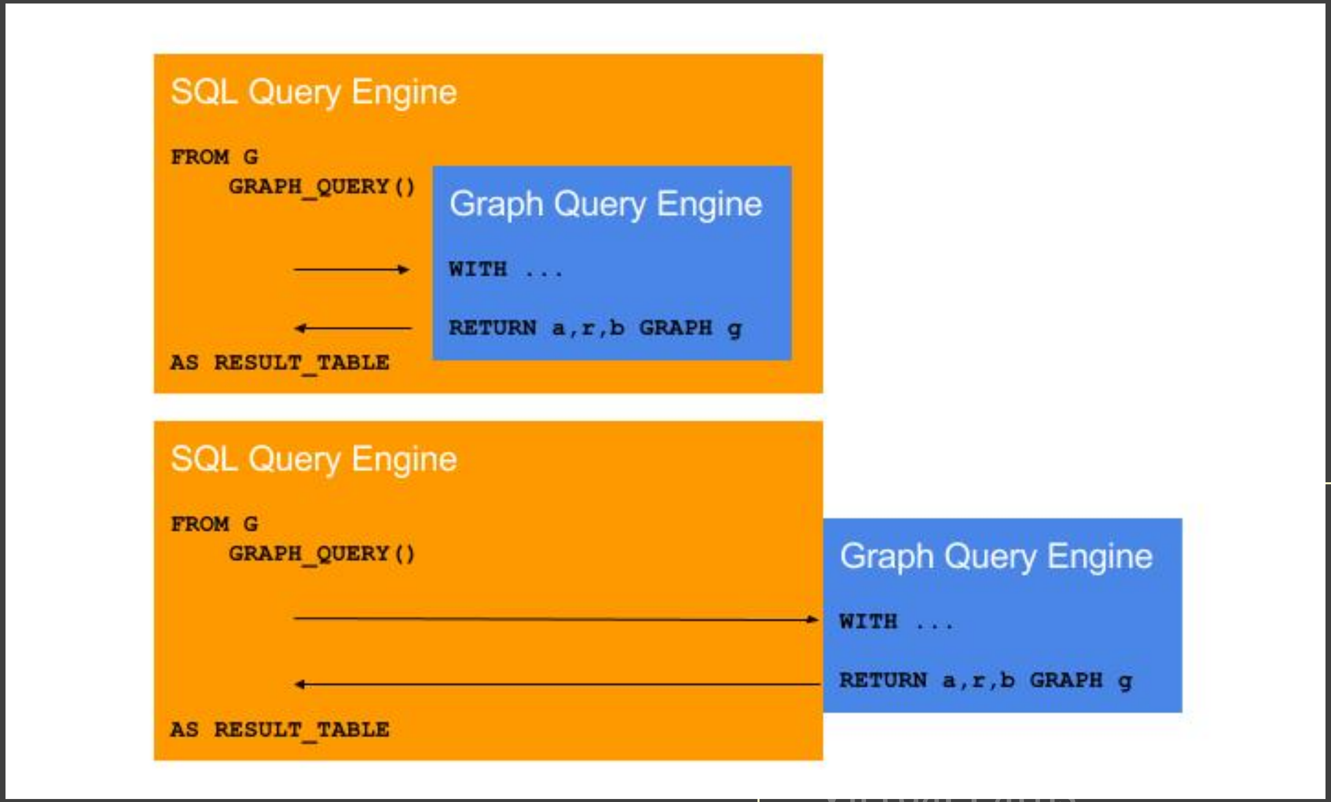
Tabular projections, graph projections, update DML

Direction is not settled

Written contributions from Oracle, Microsoft and Neo4j

SAP, Teradata, IBM are also participants

Liaison with LDBC



“Morphism” + tractability

Schema constraints

SQL

openCypher

2015
Neo announced intention to make Cypher into an open standard

2016
Apache-licensed Grammar, TCK

2017
openCypher Implementers Meetings (SAP Walldorf, London) F2F + virtual meetings each month

November
Third F2F this year is discussing compositional queries and MG: collocated with DBA in France

Governance by consensus
Open to all
Implementers have strongest voice

Thank you.

Questions or comments?

HPTS Submission for this talk

In 1983 Ronald Fagin described the relational data model as a hyper-graph. But relational databases, in one sense, came from the need or desire to link files together, and to carry out set-level operations in place of network navigation. The emphasis of relational query languages was on tables first, and joining second: subliminally the relationships expressed in PK-FK pairs became inferior, run-time concerns. The entity-relationship model became occluded, in storage and in querying.

In the last five years the "property graph" data model has catapulted a graph view of linked data back into the limelight, 40 or so years on from CODASYL.

In this model (which is not fundamentally different from RDF, but has proved to be much more tractable and less encumbered for practitioners) relationships (edges) are as important as nodes (vertices). Graph path matching and traversals, at speed and for high volumes, have also proved to be critical query components.

There is a lot of current work on declarative graph query languages: the goal is an "SQL of graph data". First-generation graph query languages like Neo's Cypher or Oracle's PGQL are lacking in one key characteristic: they are not composable. SQL allows queries to operate over relations and to return relations. But today's graph queries cannot return a graph, and a view cannot frame a sub-graph on an underlying graph.

There are many other interesting issues raised by graph-centric querying. How much (back to Fagin) is graph querying just a recasting of relational processing? How might a full-power graph query language relate to SQL?

As more and more vendors release property graph features (e.g. MongoDB or SQLServer 2017, to take two recent examples), and with an explosion of research on topics such as graph representations, partitioning and query optimizations (see EmptyHeaded), it seems like graph query languages, both for reading and writing, are going to be as important as SQL or QUEL were in their day.