



Property graphs vs Semantic Graph Databases

July 2014

Forrester: what is the difference between a property graph and semantic graph database?

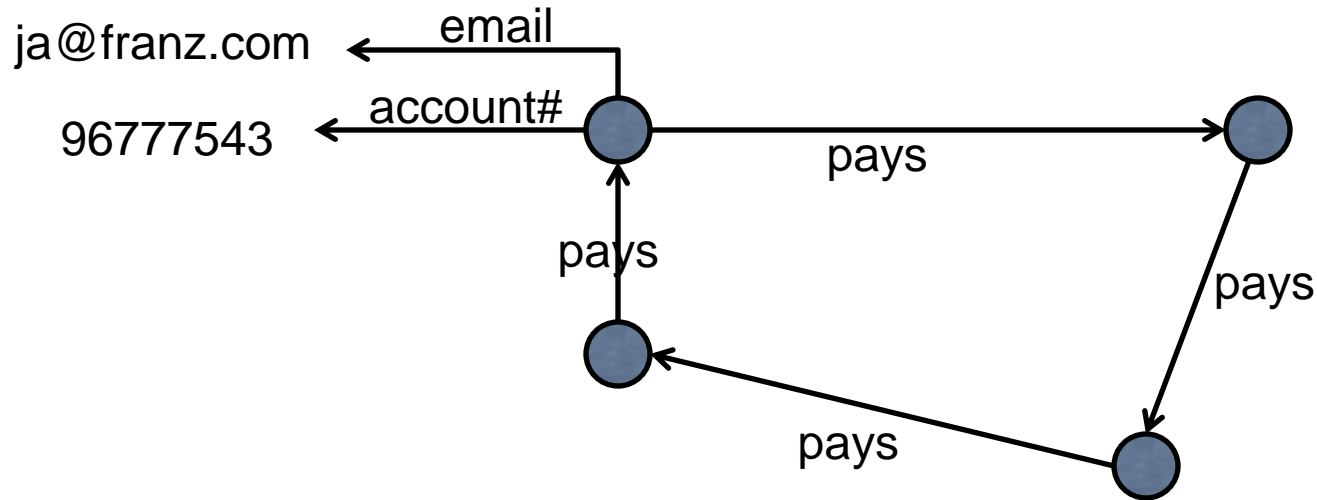
Property Graphs	Semantic Graph Database
Emerging Ad Hoc Standard	W3C standard
Schema based: need to define all types, properties, values upfront	Dynamic schema or schema-less
Property graphs limited to attributes only	Property graphs generalized to any depth
Very good at solving point solutions (ie, shortest path, graph traversal)	Optimized for both aggregated queries AND 'pointer-chasing'
Query language still in development. Cypher can't do 'or' or 'negation', query optimization still by hand	Full query language (SPARQL, W3C) equivalent to SQL, with query optimizers
Avoid long strings	Optimized to deal with arbitrary length strings (URLS)
Need effort to share disparate data	Built to link disparate data
Need to write procedural Java code	Built for rules/reasoning
Very hard to read Linked Open Data	Linked Open Data

Just to be sure:

- Semantic Graph Databases:
 - Every triple store is a SGD
- Property graph databases
 - Graph database that don't do Semantics
 - Where links can have properties themselves.
- Every day we get this question:
 - So your triple store is not a graph database because you can't do property graphs?

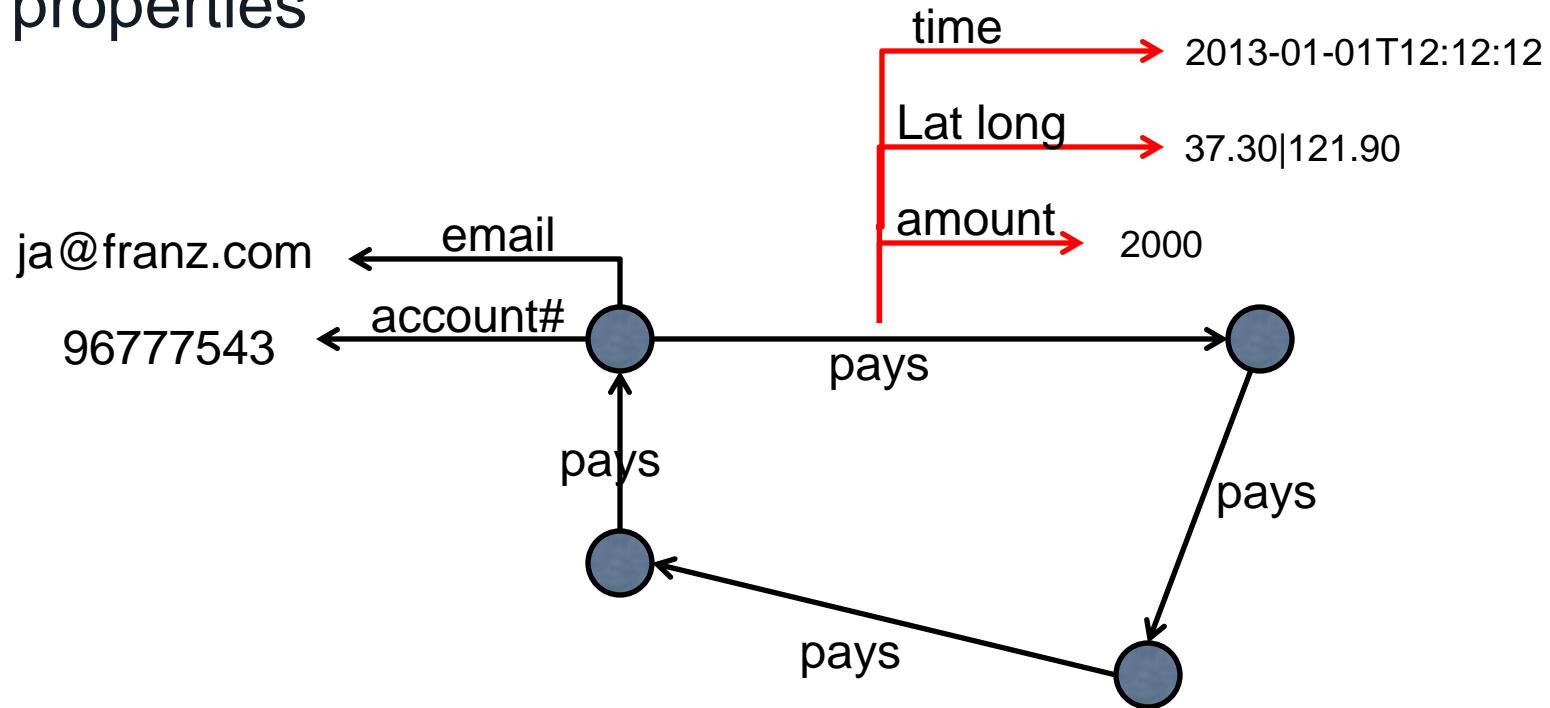
Property graphs and hypergraphs

In a property graph both nodes and links can have properties



Property graphs and hypergraphs

In a property graph both nodes and links can have properties



So: can Semantic Graph Databases NOT do property graphs?

- Yes: we can and we have a much more general mechanism

A Tale of Two Graphs: Property Graphs as RDF in Oracle

Souripriya Das

Jagannathan Srinivasan

Matthew Perry

Eugene Inseok Chong

Jayanta Banerjee

Oracle

One Oracle Drive, Nashua, NH 03062

firstname.lastname@oracle.com

ABSTRACT

Graph Databases are gaining popularity, owing to pervasiveness of graph data in social networks, physical sciences, networking, and web applications. A majority of these databases are based on the property graph model, which is characterized as key/value-based, directed, and multi-relational. In this paper, we consider the problem of supporting property graphs as RDF in Oracle Database. We introduce a property graph to RDF transformation scheme. The main challenge lies in representing the key/value properties of property graph edges in RDF. We propose three models: 1) named graph based, 2) subproperty based, and 3) (extended) reification based, all of which can be supported with RDF capabilities in Oracle Database. These models are evaluated with respect to ease of SPARQL query formulation, join complexities, skewness in generated RDF data, query performance, and storage overhead. An experimental study with a real-life Twitter social network dataset on Oracle Database 12c demonstrates the feasibility of representing property graphs as RDF and presents a quantitative performance comparison of the proposed models.

Categories and Subject Descriptors

H.2.m [Database Management]: Miscellaneous

General Terms

Algorithms, Performance, Design, Experimentation, Theory.

Keywords

Property Graph, RDF, SPARQL, Graph Database, Social Network.

1. INTRODUCTION

Graph Databases are gaining popularity, owing to pervasiveness of graph data in social networks, physical sciences, networking, and web applications. A majority of these databases (such as Neo4j [6], DEX [7], InfiniteGraph [8]) are based on the *property graph* model [2].

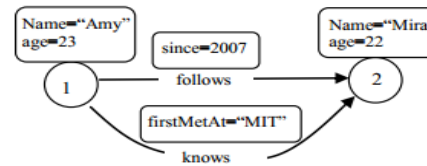


Figure 1. A sample property graph.

Property graph data is typically accessed through the de facto standard Blueprints Java API [3] or some proprietary query language. The query languages over property graphs (such as Cypher [14]) have typically focused on finding paths once the start node, or qualifying start nodes identified with certain key/values are specified. The edges themselves can be traversed by considering the associated key/value pairs. Throughout the paper, we use the words *node* and *vertex* interchangeably.

In contrast, RDF [1] provides a way of specifying directed, labeled graphs. Each directed, labeled edge is represented by a triple: $\langle \text{subject}, \text{predicate}, \text{object} \rangle$ where the predicate is the label for a directed edge from the subject node to the object node. The use of quads, instead of triples, is becoming popular in practice and is included in the new W3C RDF1.1 Recommendation [33]. A quad extends a triple by allowing an optional *named graph* component and is represented as: $\langle \text{subject}, \text{predicate}, \text{object}, \text{graph} \rangle$. Each of the components of a triple or a quad must be an RDF term. RDF terms can be of three types: Internationalized Resource Identifier (IRI), blank node, or literal. Restrictions on types of RDF terms that can be used in a component position are: 1) subject must be an IRI or a blank node; 2) predicate must be an IRI; 3) object must be an IRI, a blank node, or a literal; 4) graph, if present, must be an IRI or a blank node.

RDF graphs are typically queried using the standard SPARQL query

At least three ways to do it...

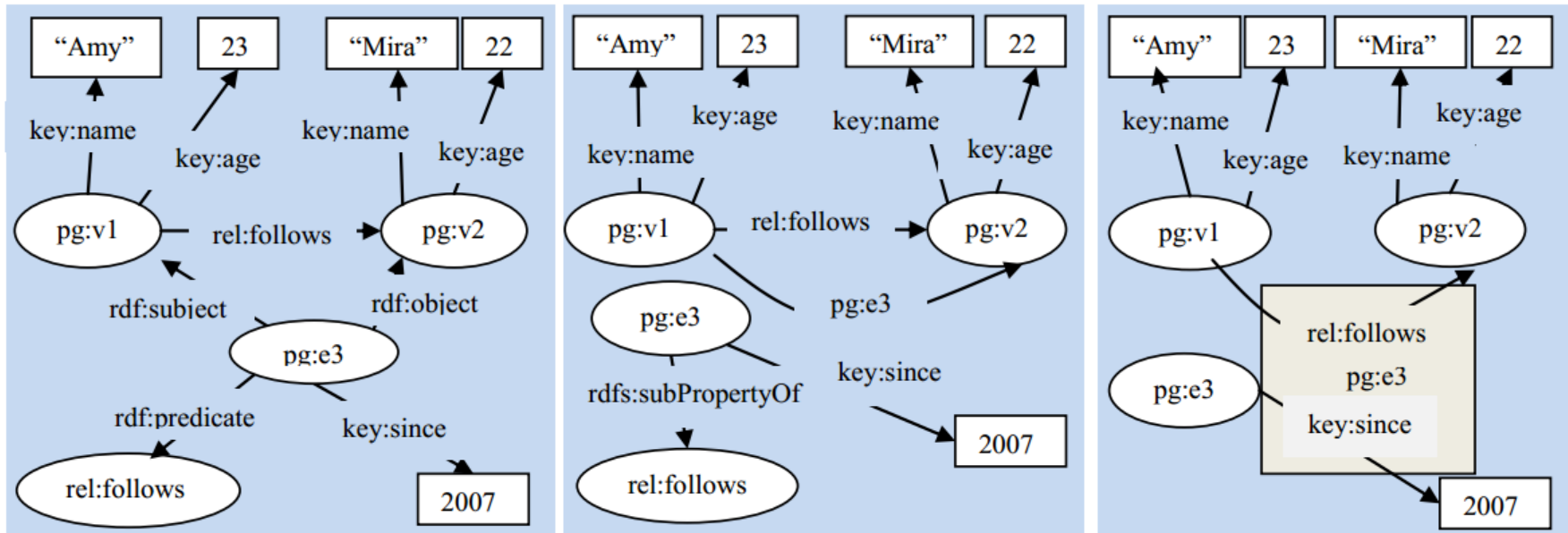


Figure 2. a) (Extended) Reification-based, b) RDF subproperty-based c) Named graph based representation of a property graph

In AllegroGraph we prefer option 3 (and option 4 😊)

- Using the graph slot of a triple
- But we can also use the triple id if you want to use the graph slot for other purposes
- And I want to demonstrate that SGD property graphs are more **flexible** and more **expressive**.

In SGD properties can have links to other objects (not just attribute/value like property graph databases)

```
:jans :weighs 105 :ph1
```

```
:ph1 :author :sophia
```

```
:ph1 :certainty 100%
```

```
:ph1 :date 2012-12-12
```

```
:jans :weighs 105 :ph2
```

```
:ph2 :author :jans
```

```
:ph2 :certainty 10%
```

```
:ph2 :date 2012-12-12
```

```
:sophia :age 47
```

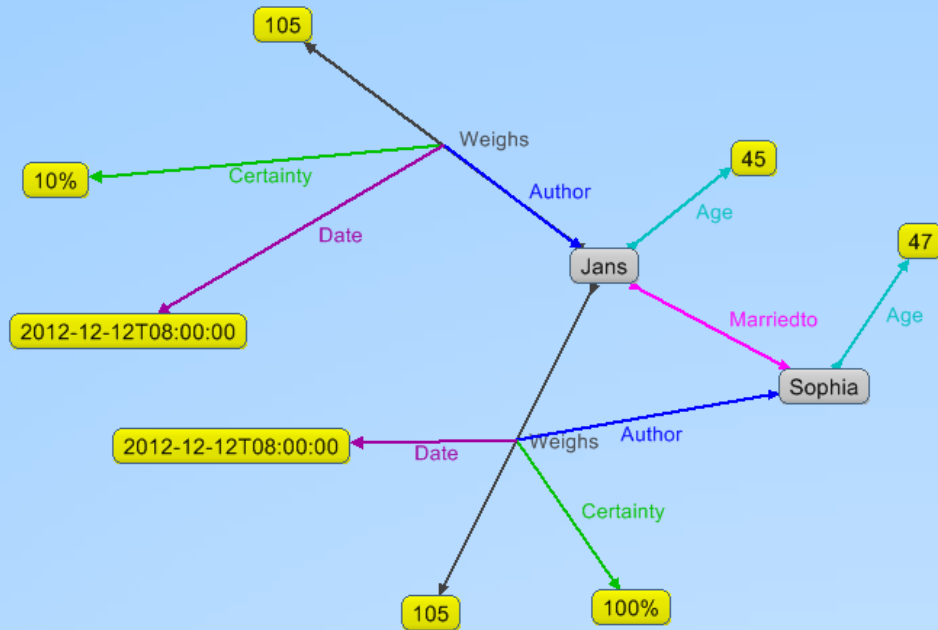
```
:jans :age 45
```

```
:jans :marriedto :sophia
```

```
:marriedto rdf:type owl:SymmetricProperty
```

- Age
- Author
- Certainty
- Date
- Marriedto
- Weighs

- Literal
- No Type



And you can use regular SPARQL or Prolog to query the property graph.

find a statement about the weight of a person where this person and his wife disagree on the certainty that he has this weight.

```
select * {  
  graph ?h1 {?person :weighs ?weight .} ?h1 :author ?person; :certainty ?c1 .  
  graph ?h2 {?person :weighs ?weight .} ?h2 :author ?a2 ; :certainty ?c2 . filter( ?c1 != ?c2 )  
  ?person :marriedto ?a2 .  
}
```

a2	c1	c2	h1	h2	person	weight
sophia	10%	100%	ph2	ph1	jans	105

```
(select (?person)  
  (a ?person :weighs ?weight (:author (?person :marriedto ?a2) :certainty ?c1))  
  (a ?person :weighs ?weight (:author ?a2 :certainty (not ?c2))))
```

Even cooler: recursively apply properties on properties

```
:jans :weighs 105 :ph1

:ph1 :author :sophia :ph3 ;; new
:ph1 :certainty 100%

:ph3 :certainty 20 ; new

:jans :weighs 105 :ph2

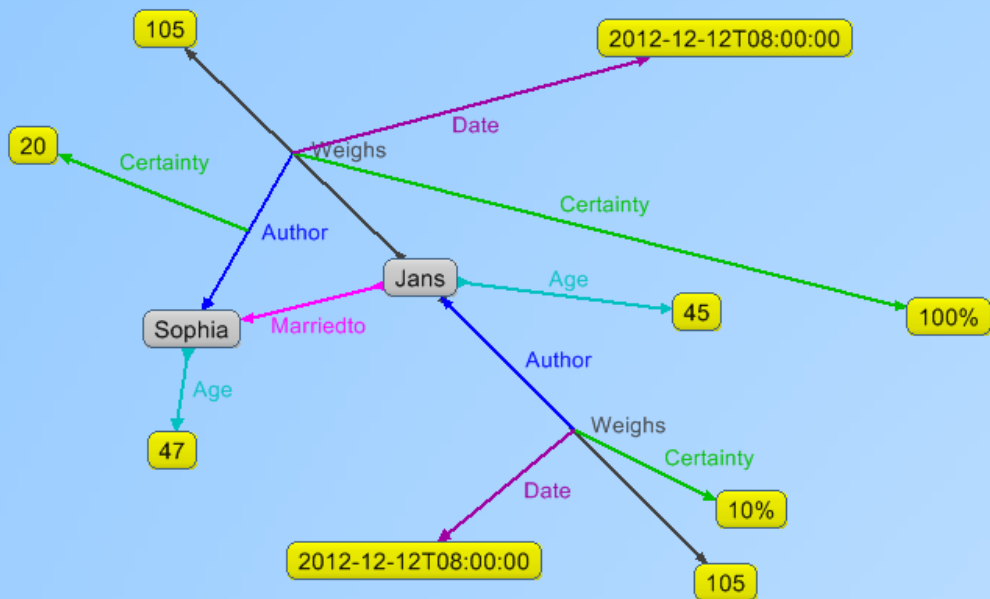
:ph2 :author :jans
:ph2 :certainty 10%

:jans :age 45
:sophia :age 47
:jans :marriedto :sophia

:marriedto rdf:type owl:SymmetricProperty
```

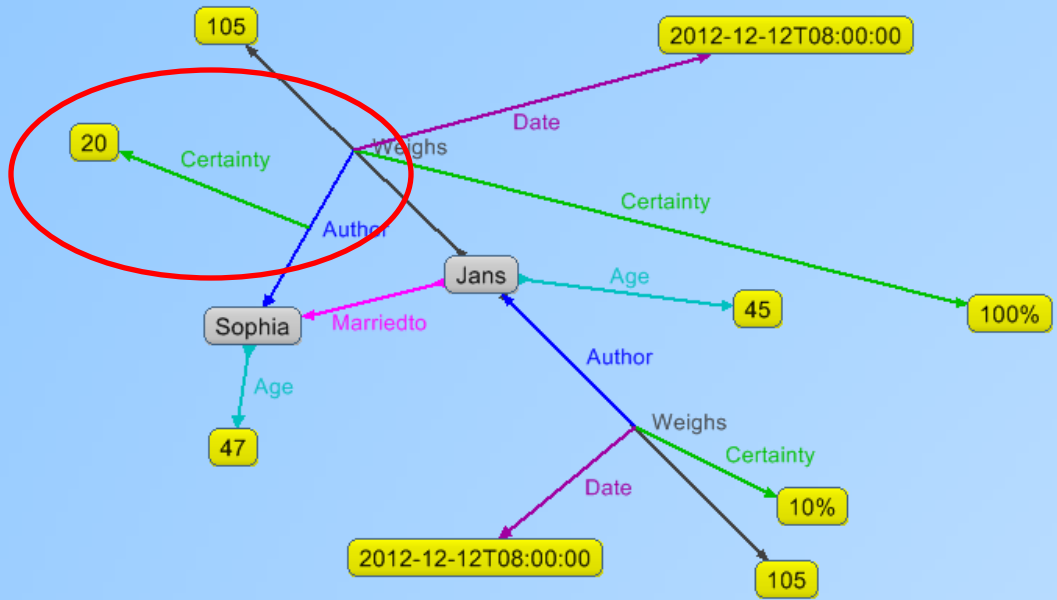
- Age
- Author
- Certainty
- Date
- Marriedto
- Weighs

- Literal
- No Type

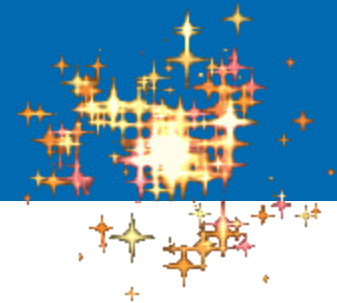


- Age
- Author
- Certainty
- Date
- Marriedto
- Weighs

- Literal
- No Type



And we happily keep SPARQLing



a person cannot believe that *his* wife said something about his weight

```
select * {
  graph ?h1 { ?person :weighs ?weight .}
  graph ?h2 { ?h1 :author ?a2 . }
  ?h2 :certainty ?c1 . }
filter ( ?c1 <= 20 )
}
```

a2	c1	h1	h2	person	weight
sophia	20	ph1	ph3	jans	105

```
(select (?person)
  (a ?person :weighs ?weight (:author ?a (:certainty (<= 20))))
  (a ?person :marriedto ?a))
```


Property graphs with triple-ids

Jans' wife says that he is 100 kilo

w3c standard reification

```
:st1 rdf:type rdf:statement
:st1 rdf:subject :jans
:st1 rdf:predicate :weighs
:st1 rdf:object :100
:st1 dc:creator :hisWife
```

using the graph

```
:jans :weighs :100 :st1
:st1 dc:creator :hisWife
```

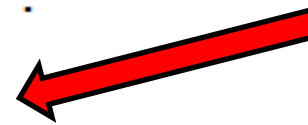
using triple id

```
:jans :weights :100
:triple1 dc:creator :hisWife
```

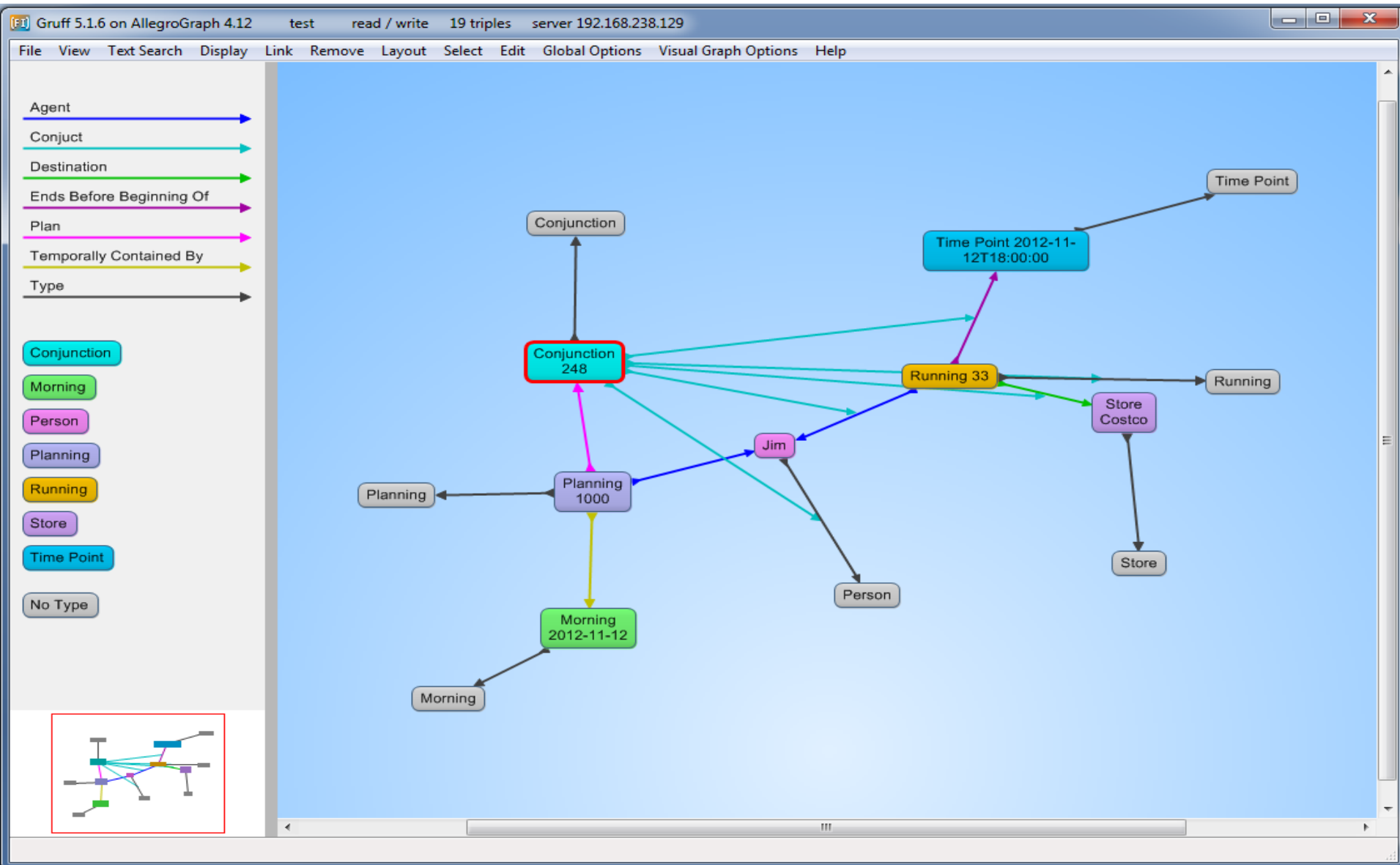
SPARQL can do this for you

```
delete {
  ?triple a rdf:Statement .
  ?triple rdf:subject ?s .
  ?triple rdf:predicate ?p .
  ?triple rdf:object ?o .
  ?triple ?property ?object .
}
insert {
  findId ?id { ?s ?p ?o }
  ?id ?property ?object .
}
where {
  ?triple a rdf:Statement .
  ?triple rdf:subject ?s .
  ?triple rdf:predicate ?p .
  ?triple rdf:object ?o .
  ?triple ?property ?object .
  filter( ! ?property in ( rdf:type, rdf:subject,
                          rdf:predicate, rdf:object ) )
}
```

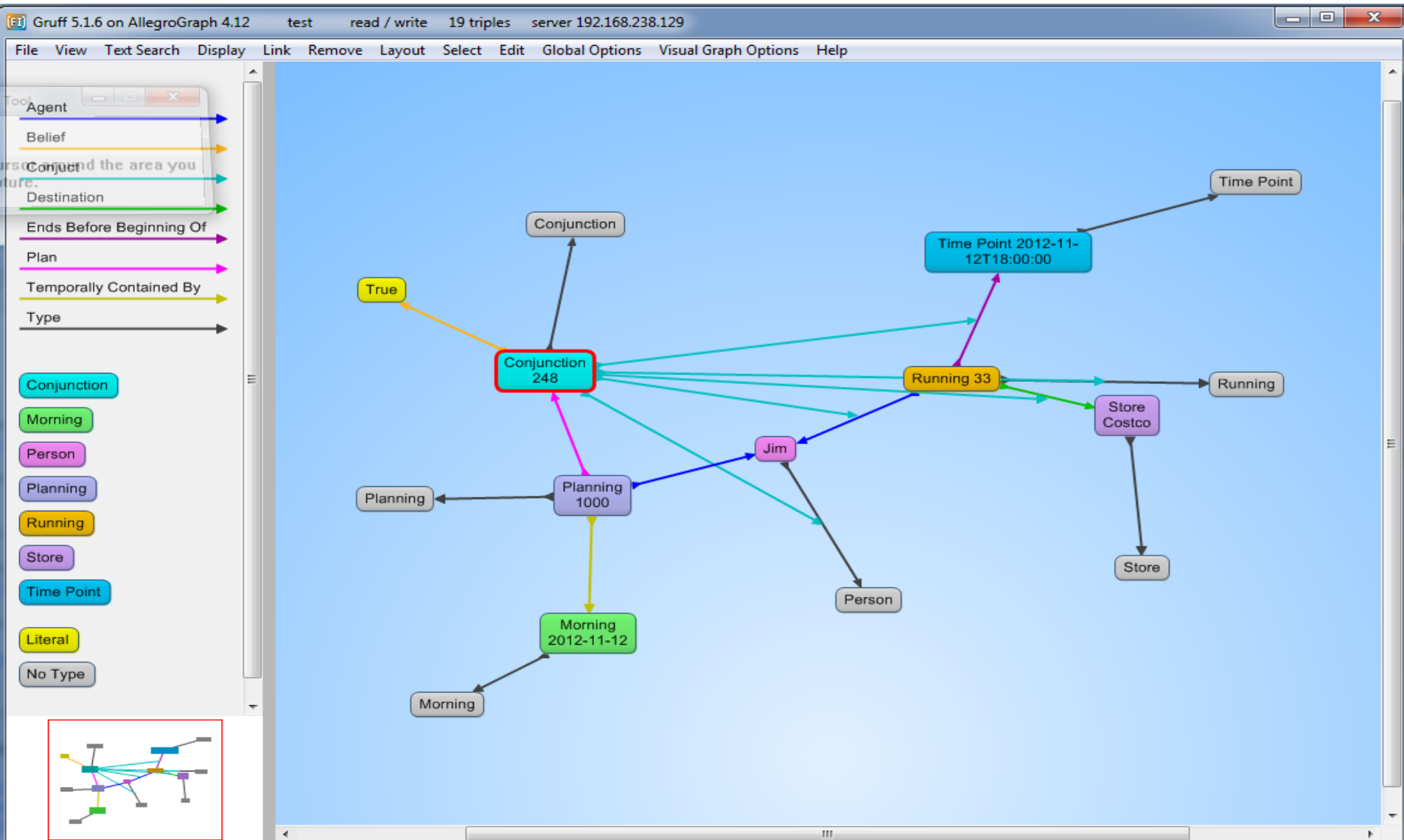
A tiny bit of magic



This morning, Jim planned on running to Costco before 6:00



This morning, Jim planned on running to Costco before 6:00



Forrester: what is the difference between a property graph and semantic graph database?

Property Graphs	Semantic Graph Database
Emerging Ad Hoc Standard	W3C standard
Schema based: need to define all types, properties, values upfront	Dynamic schema or schema-less
Property graphs limited to attributes only	Property graphs generalized to any depth
Very good at solving point solutions (ie, shortest path, graph traversal)	Optimized for both aggregated queries AND 'pointer-chasing'
Query language still in development. Cypher can't do 'or' or 'negation', query optimization still by hand	Full query language (SPARQL, W3C) equivalent to SQL, with query optimizers
Avoid long strings	Optimized to deal with arbitrary length strings (URLS)
Need effort to share disparate data	Built to link disparate data
Need to write procedural Java code	Built for rules/reasoning
Very hard to read Linked Open Data	Linked Open Data

The 'Obama' Query in English

“Find the median income of the area where Barack Obama was born.”

- Find birth place of Obama in DBPedia
- Find GeoNamesID for that place
- Goto GeoNames to find all the towns within 10 miles of that ID (using geospatial query) that are of type 'populated area' -> list of GeoNameIDs
- Find the Census place that are the same as the GeoNameIDs
- Find the Census tables for the Census places
- Find the medianIncomeAbove15yearsOfAge for each table (for each place)
- Report

Catalogs

- [system](#)


Repositories

- [census](#) ×
- [census-geonames-links-test](#) ×
- [dbpedia](#) ×
- [geonames](#) ×
- [test](#) ×
- [test2](#) ×

Create new repository

- Name:
- [Restore from a backup](#)

Start session

Session specification:  (autocommit, load initfile)

Running sessions:

- [Federated store](#) (port 52643)
- [Federated store](#) (port 46506)

Site settings

There are 2 [user accounts](#). Anonymous access is allowed, but regular users can not create new accounts. [[allow](#)]

Google Maps key:

(When there is no key set, mapping is unavailable. You can obtain a key from [Google](#).)

Edit the site's [initfile](#).

Edit query

Query language: Query planner: Result limit: [show my namespaces, add a namespace](#)

```
1 PREFIX geo: <http://franz.com/ns/allegrograph/3.0/geospatial/>
2 PREFIX geonames: <http://sws.geonames.org/>
3 PREFIX dbpedia_rsrc: <http://dbpedia.org/resource/>
4 PREFIX dbpedia_onto: <http://dbpedia.org/ontology/>
5 PREFIX dbpedia_prop: <http://dbpedia.org/property/>
6 PREFIX census: <tag:govshare.info,2005:rdf/census/>
7 PREFIX census_samp: <tag:govshare.info,2005:rdf/census/details/samp/>
8
9
10 SELECT ?censusplace ?income {
11   dbpedia_rsrc:Barack_Obama dbpedia_onto:birthPlace ?birthplace .
12   ?birthplace dbpedia_prop:hasGeonamesID ?geonamesresource .
13
14   SERVICE <http://blade8:10001/repositories/geonames>
15     { ?geonamesresource geonames:isAt5 ?location .
16
17       ?otherplace geo:inCircleMiles (geonames:isAt5 ?location 10) .
18       ?otherplace geonames:feature_code "PPL" .
19       ?geonamesresource geonames:feature_code "PPL" .
20
21     SERVICE <http://blade8:10001/repositories/census>
22       { ?censusplace dbpedia_prop:hasGeonamesID ?otherplace .
23
24         ?censusplace census:details ?detail .
25         ?detail census_samp:population15YearsAndOverWithIncomeIn1999 ?d .
26         ?d census_samp:medianIncomeIn1999 ?income .
27       }
28     }
29 }
30
```

[add Prolog functors](#)

Edit query

Query language: SPARQL ▾ Query planner: default ▾ Result limit: 100 ▾ [show my namespaces, add a namespace](#)

```

1 PREFIX geo: <http://franz.com/ns/allegrograph/3.0/geospatial/>
2 PREFIX geonames: <http://sws.geonames.org/>
3 PREFIX dbpedia_rsrc: <http://dbpedia.org/resource/>
4 PREFIX dbpedia_onto: <http://dbpedia.org/ontology/>
5 PREFIX dbpedia_prop: <http://dbpedia.org/property/>
6 PREFIX census: <tag:govshare.info,2005:rdf/census/>
7 PREFIX census_samp: <tag:govshare.info,2005:rdf/census/details/samp/>
8
9
10 SELECT ?censusplace ?income {
11   dbpedia_rsrc:Barack_Obama dbpedia_onto:birthPlace ?birthplace .
12   ?birthplace dbpedia_prop:hasGeonamesID ?geonamesresource .
13
14   SERVICE <http://blade8:10001/repositories/geonames>
15     { ?geonamesresource geonames:isAt5 ?location .
16
17       ?otherplace geo:inCircleMiles (geonames:isAt5 ?location 10) .
18       ?otherplace geonames:feature_code "PPL" .
19       ?geonamesresource geonames:feature_code "PPL" .
20
21     SERVICE <http://blade8:10001/repositories/census>
22       { ?censusplace dbpedia_prop:hasGeonamesID ?otherplace .
23
24         ?censusplace census:details ?detail .
25         ?detail census_samp:population15YearsAndOverWithIncomeIn1999 ?d .
26         ?d census_samp:medianIncomeIn1999 ?income .
27       }
28     }
29 }

```

Execute

[add Prolog functors](#)

Result

Download as SPARQL JSON ▾

censusplace	income
iroquois_point	"33005"
pearl_city	"25776"
waimalu	"34777"
maunawili	"41886"

Statements with [waimalu](#) » as the subject.

Predicate	Object
hasGeonamesID	5854530/
dc:title	"Waimalu"
dcterms:isPartOf	ewa
households	"10999"
waterArea	"525475 m^2"
population	"29371"
lat	"21.39915"
rdf:type	Village
long	"-157.948432"
landArea	"15299516 m^2"
details	censustables

Statements with [waimalu](#) » as the object.

Subject	Predicate
ewa	dcterms:hasPart

Statements with `consustables` » as the subject.

Predicate	Object
totalPopulation	:b8298E448x70808955
housingUnits	:b8298E448x47273736
housingUnits	:b8298E448x47273843
civilianNoninstitutionalizedPopulation5YearsAndOver	:b8298E448x46754316
civilianNoninstitutionalizedPopulation5YearsAndOver	:b8298E448x46754452
nonrelatives	:b8298E448x42063463
population5YearsAndOverInHouseholds	:b8298E448x53952390
totalPopulation	:b8298E448x53952404
specifiedRenteroccupiedHousingUnits	:b8298E448x47273962
specifiedRenteroccupiedHousingUnits	:b8298E448x47274076
specifiedRenteroccupiedHousingUnits	:b8298E448x47274173
specifiedRenteroccupiedHousingUnits	:b8298E448x47274179
specifiedRenteroccupiedHousingUnits	:b8298E448x71518831
population30YearsAndOverInHouseholds	:b8298E448x53952072
population5YearsAndOver	:b8298E448x53952176
population5YearsAndOver	:b8298E448x53952284
occupiedHousingUnits	:b8298E448x47273725
occupiedHousingUnits	:b8298E448x47273731
occupiedHousingUnits	:b8298E448x47273760
occupiedHousingUnits	:b8298E448x47273836
occupiedHousingUnits	:b8298E448x47273863
occupiedHousingUnits	:b8298E448x71518926
occupiedHousingUnits	:b8298E448x71518992
populationInGroupQuarters	:b8298E448x42063575
populationInGroupQuarters	:b8298E448x42063746
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47273883
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47273916
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47273970
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47273974
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47273980
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47273984
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47274081
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47274085
specifiedRenteroccupiedHousingUnitsPayingCashRent	:b8298E448x47274089

Forrester: what is the difference between a property graph and semantic graph database?

Property Graphs	Semantic Graph Database
Emerging Ad Hoc Standard	W3C standard
Schema based: need to define all types, properties, values upfront	Dynamic schema or schema-less
Property graphs limited to attributes	Property graphs generalized to any depth
Very good at solving point solutions (shortest path, graph traversal)	Optimized for both aggregated queries AND 'r-chasing'
Query language still in development Cypher can't do 'or' or 'negation', query optimization still by hand	Query language (SPARQL, W3C) Equivalent to SQL, with query optimizers
Avoid long strings	Optimized to deal with arbitrary length strings
Need effort to share disparate data	Built to link disparate data
Need to write procedural Java code	Built for rules/reasoning
Very hard to read Linked Open Data	Linked Open Data



The End

