Knowledge Graph Conference 2020

Modeling Evolving Data in Graphs While Preserving Backward Compatibility: The Power of RDF Quads

Souri Das, Ph.D., Architect
Matt Perry, Ph.D., Consultant Member of Technical Staff
Eugene Chong, Ph.D., Consultant Member of Technical Staff

Oracle Server Technologies

May 05, 2020

Souripriya Das (Souri)

https://www.linkedin.com/in/souripriya-souri-das-ph-d-48801911/

Architect at Oracle
- RDF Knowledge Graph
- Property Graph

Education
- Ph.D., Rutgers University
- M.S., Vanderbilt University
- B.Tech., Indian Institute of Technology (IIT), Kharagpur

Standards Activity
- W3C SPARQL 1.0 and 1.1
- W3C RDB2RDF, Editor of R2RML

Publications in SW and Database Area
- ICDE, EDBT, VLDB, CIKM
- Patents in Database and Graph technologies

Matthew Perry

Engineer at Oracle
- RDF Knowledge Graph
- PGQL on RDBMS

Ph.D. in Computer Science
- Wright State University
- Geospatial Semantic Web Area

Standards Activity
- W3C SPARQL 1.1 Working Group
- OGC GeoSPARQL

Papers in SW and Database Area
- ICDE, EDBT, ACM-GIS
- Terra Cognita workshop series

Eugene Inseok Chong

Consulting MTS at Oracle
Working on Graph Databases
Developer of Oracle Index Organized Tables, Reference Partitioned Tables, 32K Varchar, and Domain Indexes

Ph.D. in CS from Northwestern Univ., Evanston, IL
MS in CS from Georgia Tech, Atlanta, GA
BS in CSE from Seoul National Univ., Seoul, Korea

21 Publications including VLDB, SIGMOD, ICDE, and EDBT
Referee for journals and conferences

Specialty in Database Query Processing and Optimization

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

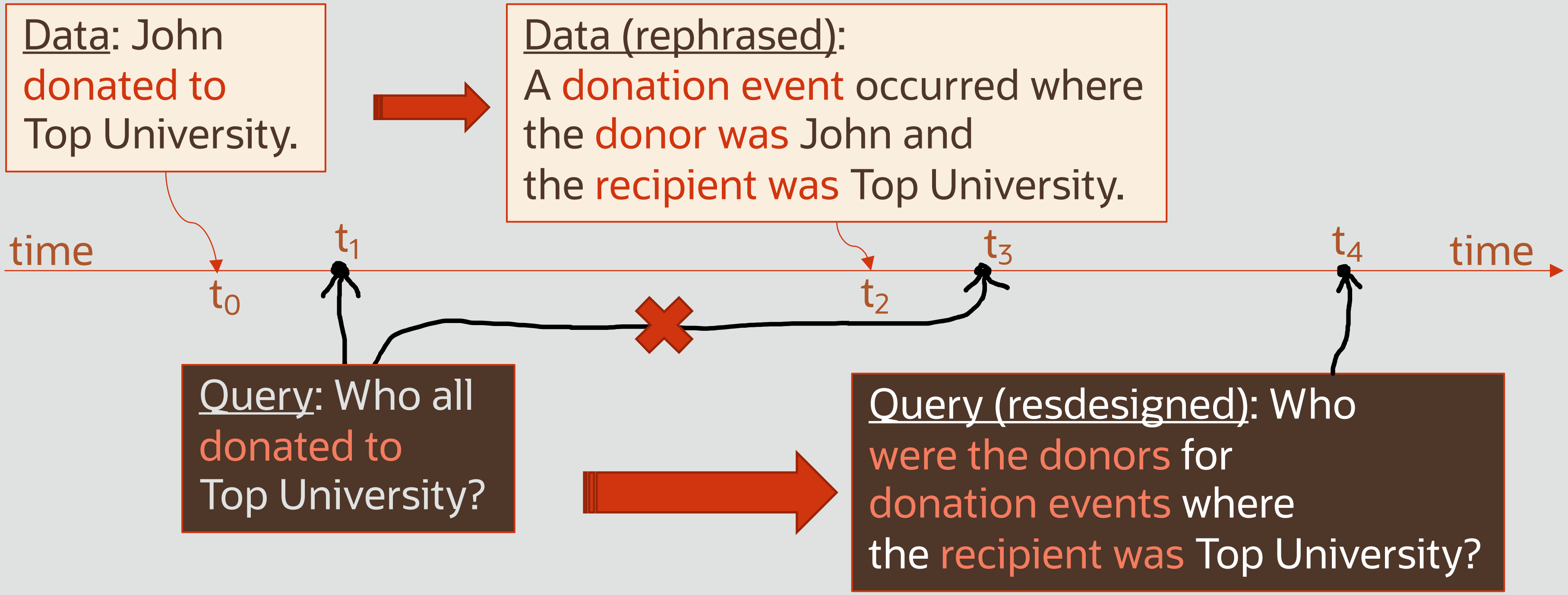
- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Backward Compatibility

validity of pre-existing queries as data evolve



Data: John donated to Top University.

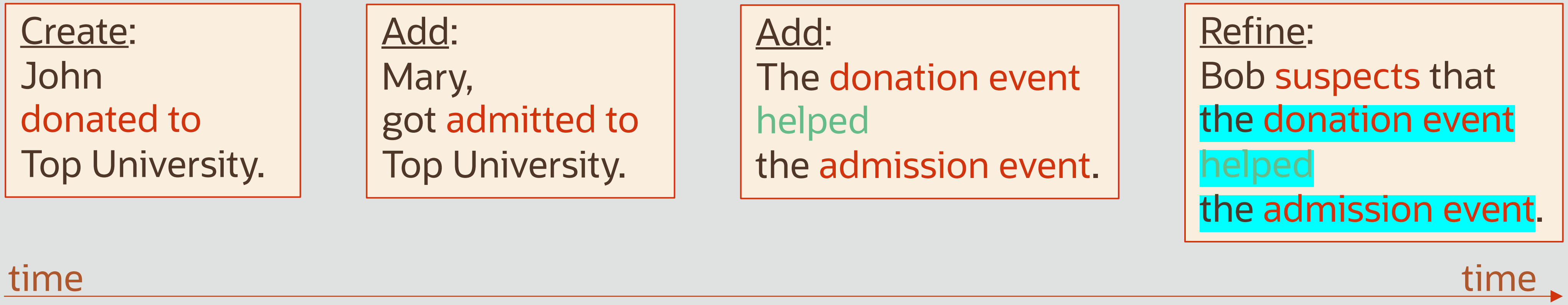
Data (rephrased): A donation event occurred where the donor was John and the recipient was Top University.

Query: Who all donated to Top University?

Query (resdesigned): Who were the donors for donation events where the recipient was Top University?

Evolving Data

Data changes are frequent, and often unanticipated



Create:
John
donated to
Top University.

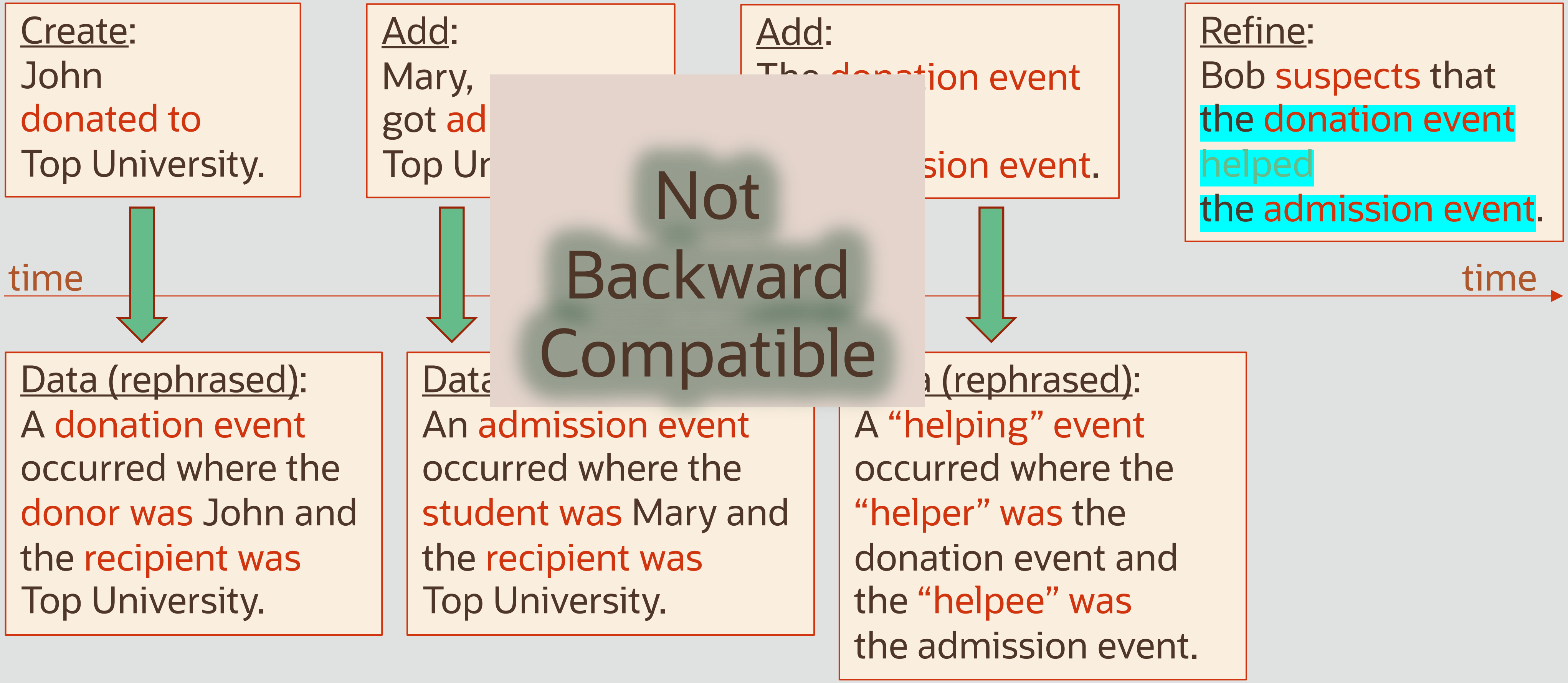
Add:
Mary,
got admitted to
Top University.

Add:
The donation event
helped
the admission event.

Refine:
Bob suspects that
the donation event
helped
the admission event.

Handling the Changes in Data

Rephrasing the data using events is one way of handling



Create:
John
donated to
Top University.

Add:
Mary,
got ad
Top Ur

Add:
The donation event
sion event.

Refine:
Bob suspects that
the donation event
helped
the admission event.

time

time

Not Backward Compatible

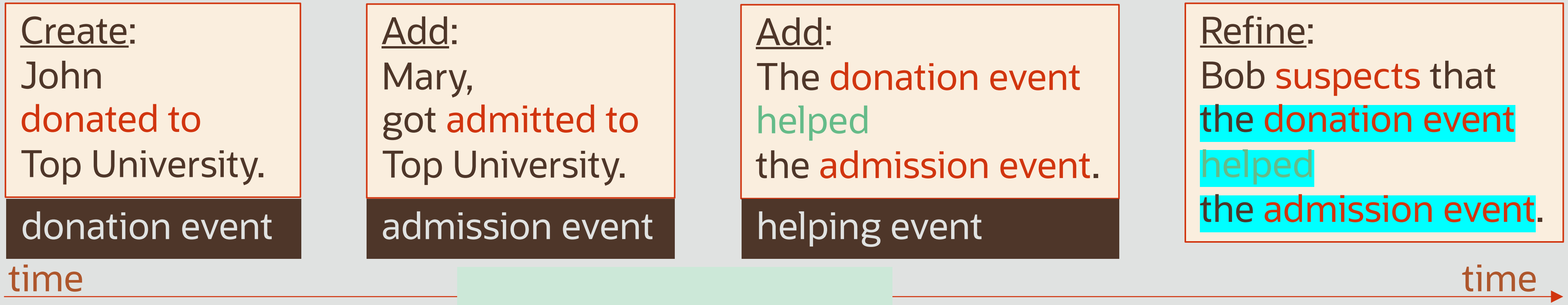
Data (rephrased):
A donation event
occurred where the
donor was John and
the recipient was
Top University.

Data
An admission event
occurred where the
student was Mary and
the recipient was
Top University.

a (rephrased):
A "helping" event
occurred where the
"helper" was the
donation event and
the "helpee" was
the admission event.

Handling the Changes in Data

Naming the events – without any rephrasing – is another way



Create:
John
donated to
Top University.

donation event

Add:
Mary,
got admitted to
Top University.

admission event

Add:
The donation event
helped
the admission event.

helping event

Refine:
Bob suspects that
the donation event
helped
the admission event.

Backward Compatible

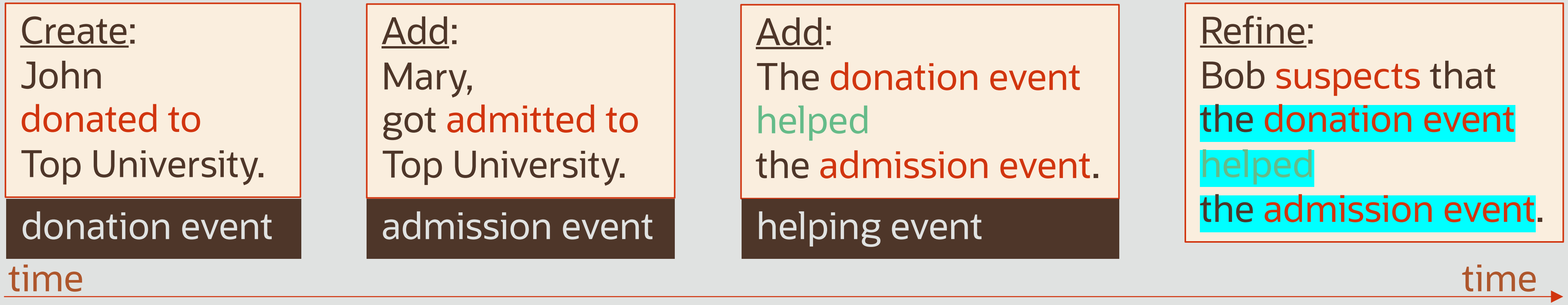
Naming is Everything !

- Name the facts.
- Compose new facts with those names.

RDF Quads

Evolving Graph: The Power of RDF Quads

Use the “graph” component to to hold the (optional) triple name



Create:
John
donated to
Top University.

donation event

Add:
Mary,
got admitted to
Top University.

admission event

Add:
The donation event
helped
the admission event.

helping event

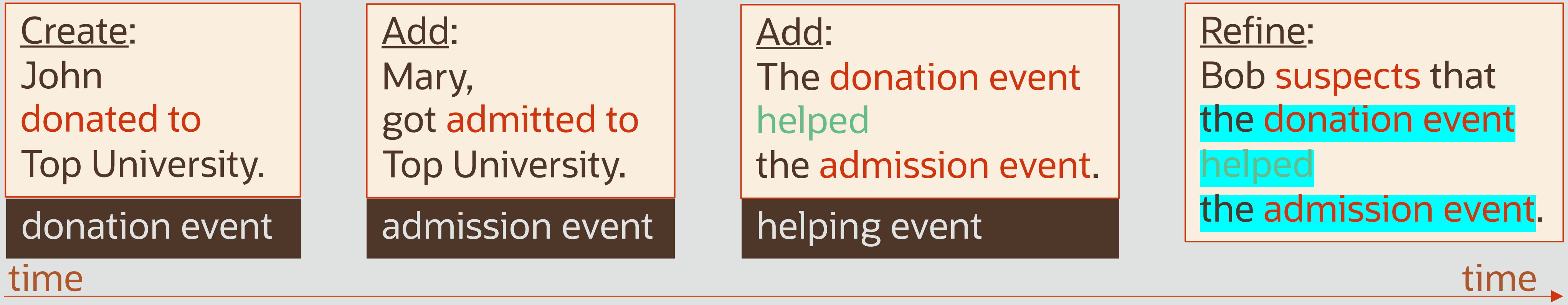
Refine:
Bob suspects that
the donation event
helped
the admission event.

graph	subject	predicate	object
:donation	:John	:donatedTo	:TopUniversity
:admission	:Mary	:admittedTo	:TopUniversity
:helping	:donation	:helped	:admission
	:Bob	:suspects	:helping

Evolving Graph: RDF# - RDF + Fact Naming

piggyback the (optional) triple name on the "predicate" component

See: https://blogs.oracle.com/oraclespatial/rdf-extending-rdf-to-support-named-triples



Create:
John
donated to
Top University.

donation event

Add:
Mary,
got admitted to
Top University.

admission event

Add:
The donation event
helped
the admission event.

helping event

Refine:
Bob suspects that
the donation event
helped
the admission event.

graph	subject	predicate	object
	:John	[:donation] :donatedTo	:TopUniversity
	:Mary	[:admission] :admittedTo	:TopUniversity
	:donation	[:helping] :helped	:admission
	:Bob	:suspects	:helping

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

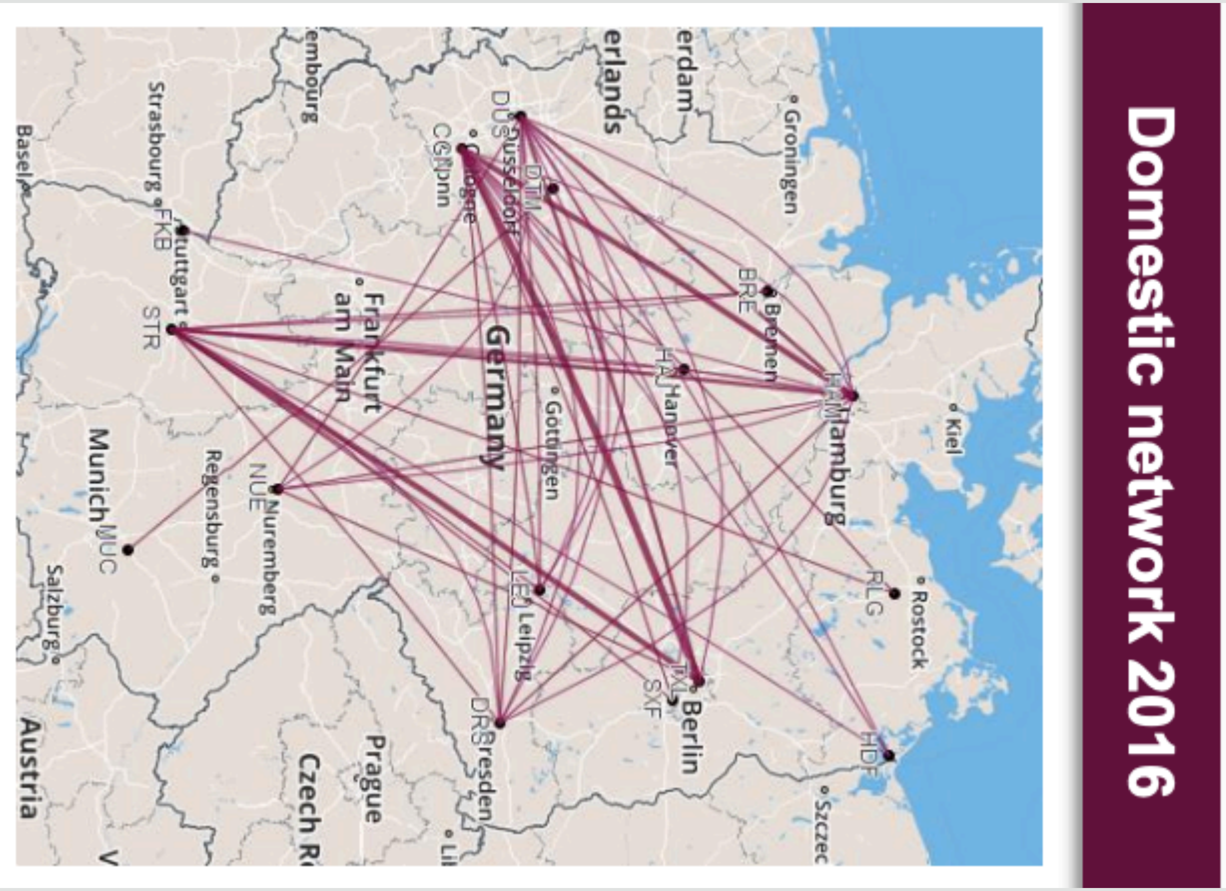
- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Regular Graphs

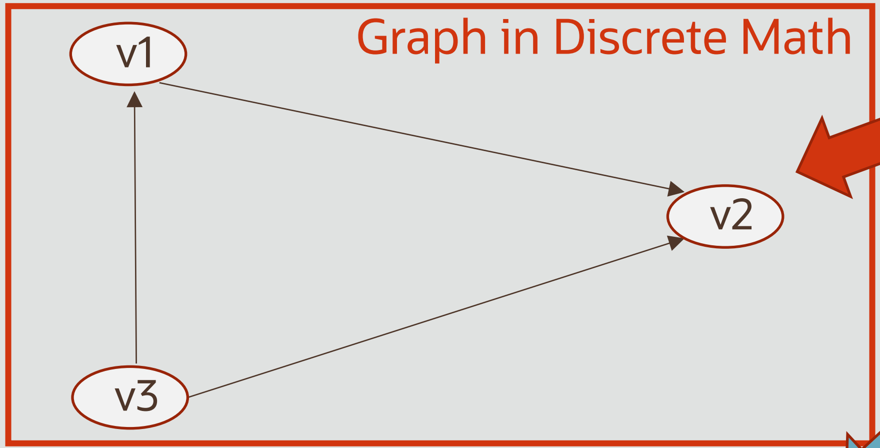
https://investor-relations.lufthansagroup.com/fileadmin/downloads/en/charts-speeches/presentations/LH-Expert-Session-Eurowings.pdf



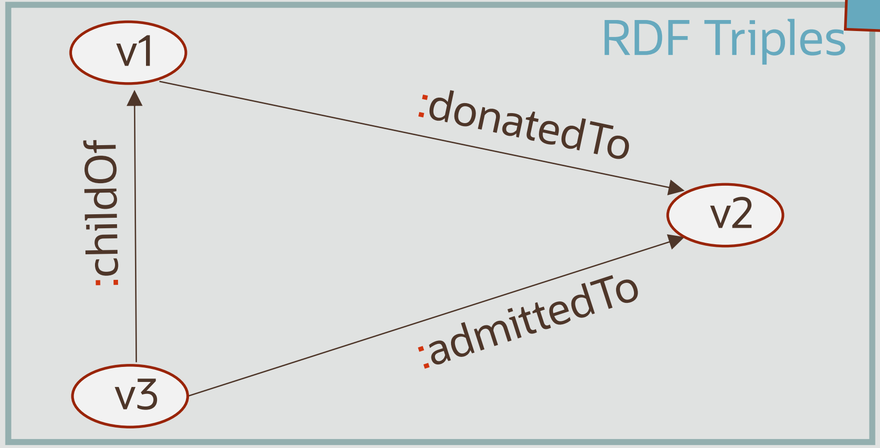
Each edge could have **distance** as an attribute.

Types of Graph

Graph in Discrete Math | RDF Triples | Property Graph | RDF Quads

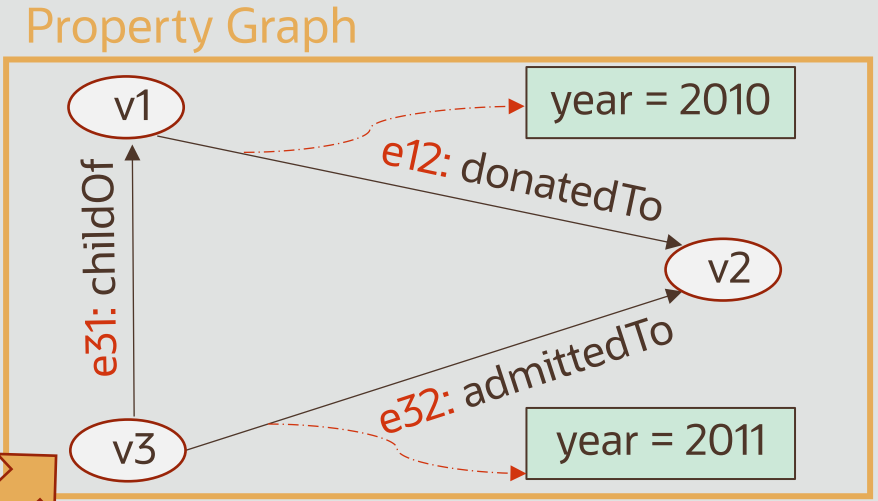


Graph in Discrete Math:
- # of relations: **1**
- for each edge
  - # attributes: **0 or 1**
  - # edges: **0**

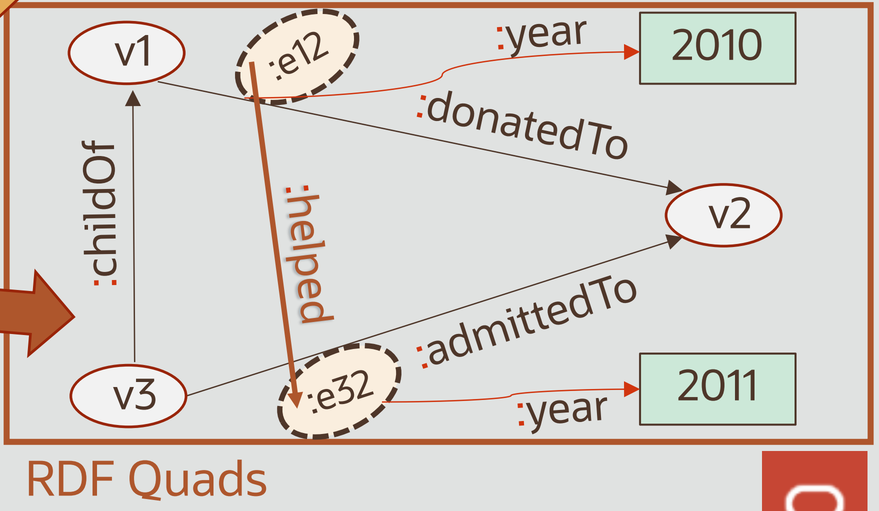


RDF Triples:
- # of relations: **many**
- for each edge
  - # attributes: **0**
  - # edges: **0**

Property Graph:
- # of relations: **many**
- for each edge
  - # attributes: **many**
  - # edges: **0**



RDF Quads:
- # of relations: **many**
- for each edge
  - # attributes: **many**
  - # edges: **many**



Types of Graph

In a Nutshell: How many edge-types (or relations) in a graph?

graph type	# of edge-types modeled in graph
graph in Math	1
RDF Triples	many
Property Graph	many
RDF Quads	many

Types of Graph

In a Nutshell: What can you hang from an edge?

graph type	# of edge-types modeled in graph	for a given edge[1] …		
		# of attributes associated with it	# of outbound edges: → vertices	# of outbound edges: → edges
graph in Math	1	0 or 1 (fixed)	-	-
RDF Triples	many	-	-	-
Property Graph	many	many	-	-
RDF Quads	many	many	many	many

[1] For RDF Quads, these apply to attribute association as well.

Comparing RDF Graph and Property Graph

Distinguishing features

	Property Graph	RDF Graph
Scope of identifiers	Local	Global (URIs)
Syntax Rules	Proprietary	Standards-based
Semantics	Embedded in application	Standard, declarative rules

Comparing RDF Graph and Property Graph

Distinguishing features

	Property Graph	RDF Graph
Vertex, Edge, Vertex-Property	Easy	Easy
Duplicate Edges	Easy	use RDF Quad
Edge-Property (KV on edge)	Easy	use RDF Quad
Multi-valued Attributes	Easy (use collection)	Easy

Comparing RDF Graph and Property Graph

Distinguishing features

	Property Graph	RDF Graph
Edge as Endpoint for Edge	“vertexify” the edge	use RDF Quad
Edge-Property as Endpoint	“vertexify” edge-property	use RDF Quad
Vertex-Property as Endpoint	“vertexify” vertex-property	use RDF Quad

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

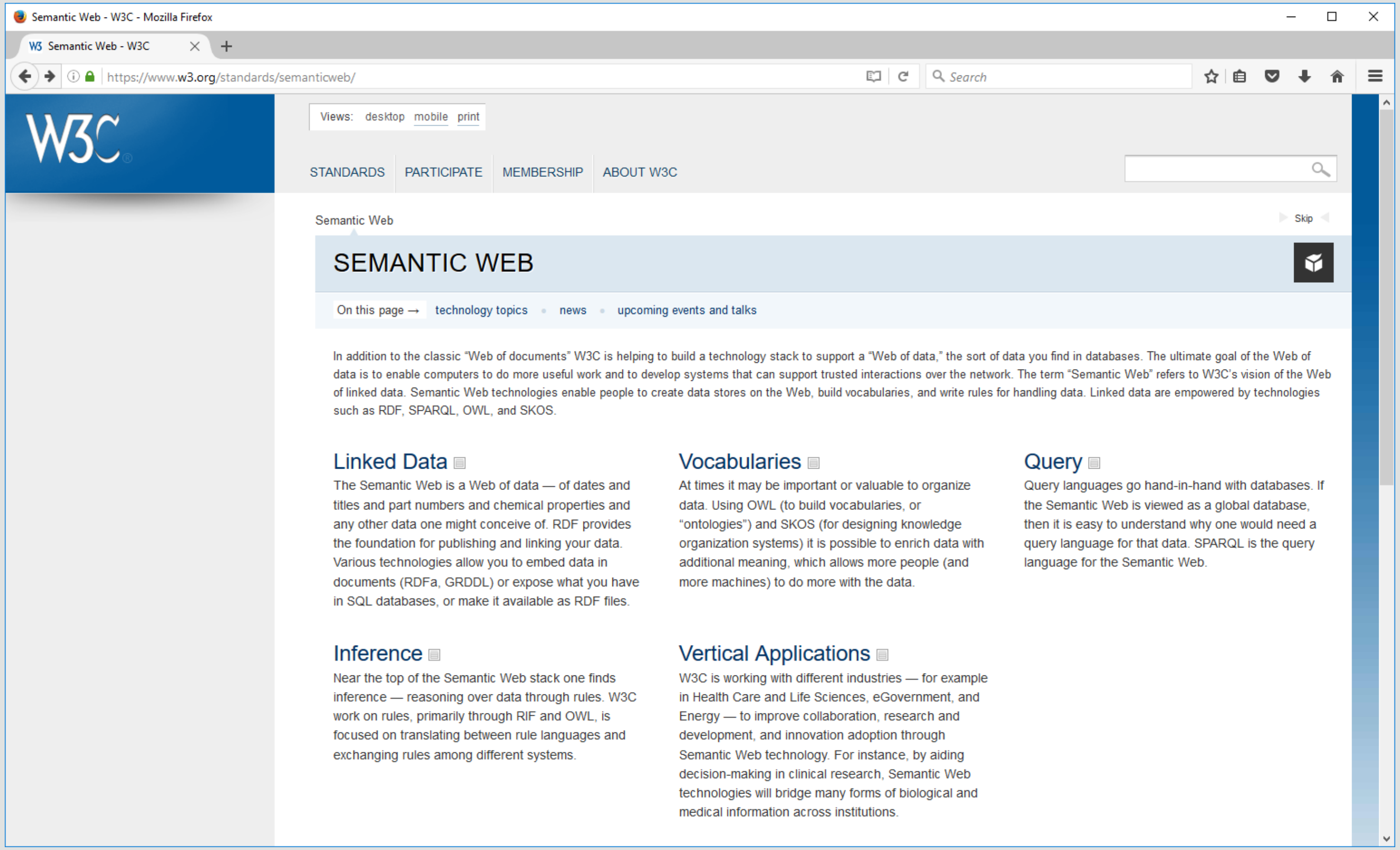
- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

W3C Standards for Knowledge Graphs

The World Wide Web Consortium has defined a suite of standards to support Linked Data and Knowledge Graphs.

Fundamental Concepts are:

- Resource Identifiers: IRIs
- Links to other resources
- Standard Data Model (RDF)
- Standard Ontology Language (OWL)
- Standard Query (SPARQL)
- Rel. Data as RDF (RDB2RDF)


Semantic Web - W3C - Mozilla Firefox
Semantic Web - W3C
https://www.w3.org/standards/semanticweb/
Search
Views: desktop mobile print
STANDARDS
PARTICIPATE
MEMBERSHIP
ABOUT W3C
Semantic Web
Skip

SEMANTIC WEB

On this page → technology topics • news • upcoming events and talks

In addition to the classic "Web of documents" W3C is helping to build a technology stack to support a "Web of data," the sort of data you find in databases. The ultimate goal of the Web of data is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network. The term "Semantic Web" refers to W3C's vision of the Web of linked data. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS.

Linked Data

The Semantic Web is a Web of data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. RDF provides the foundation for publishing and linking your data. Various technologies allow you to embed data in documents (RDFa, GRDDL) or expose what you have in SQL databases, or make it available as RDF files.

Vocabularies

At times it may be important or valuable to organize data. Using OWL (to build vocabularies, or "ontologies") and SKOS (for designing knowledge organization systems) it is possible to enrich data with additional meaning, which allows more people (and more machines) to do more with the data.

Query

Query languages go hand-in-hand with databases. If the Semantic Web is viewed as a global database, then it is easy to understand why one would need a query language for that data. SPARQL is the query language for the Semantic Web.

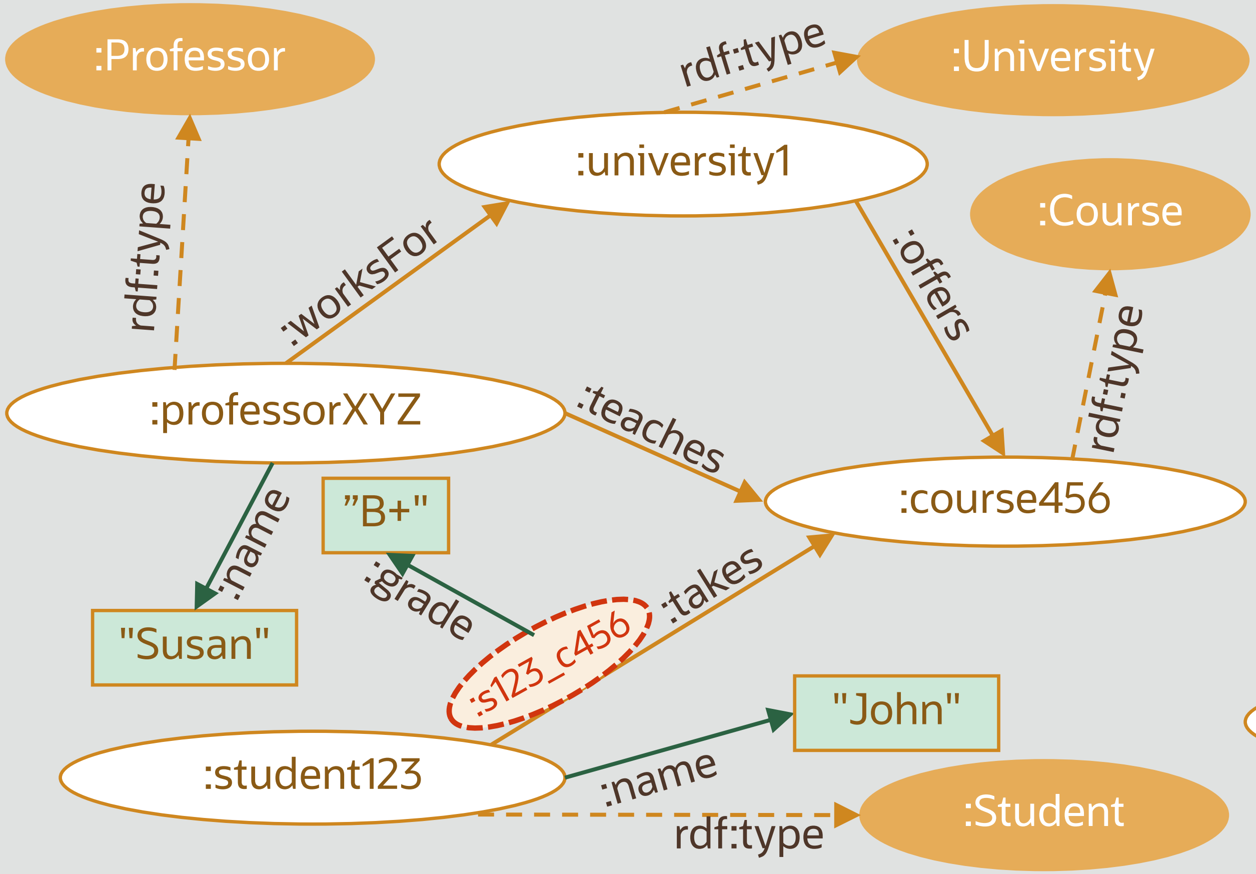
Inference

Near the top of the Semantic Web stack one finds inference — reasoning over data through rules. W3C work on rules, primarily through RIF and OWL, is focused on translating between rule languages and exchanging rules among different systems.

Vertical Applications

W3C is working with different industries — for example in Health Care and Life Sciences, eGovernment, and Energy — to improve collaboration, research and development, and innovation adoption through Semantic Web technology. For instance, by aiding decision-making in clinical research, Semantic Web technologies will bridge many forms of biological and medical information across institutions.

What is Resource Description Framework (RDF)



URI prefix and prefixed name

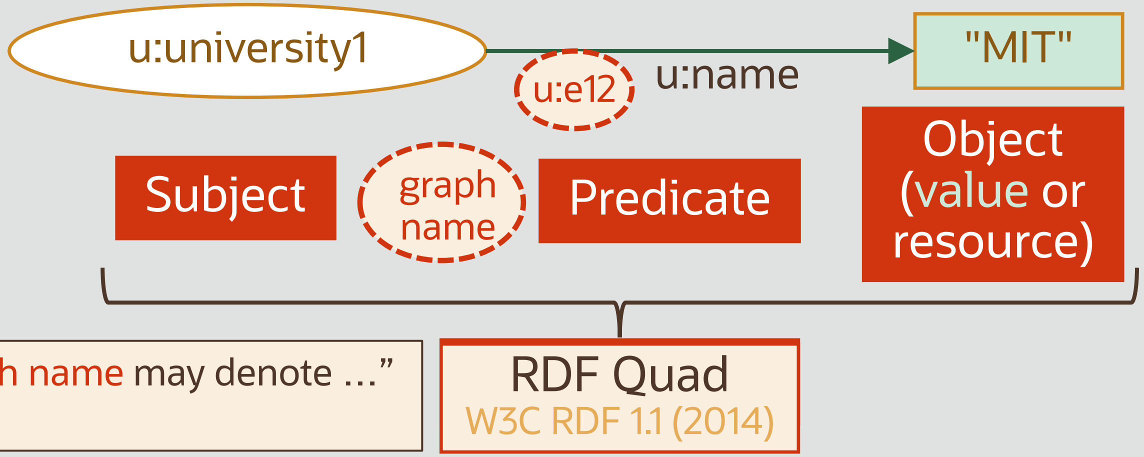
@prefix : <http://univ.org#>

An RDF graph is a directed, labeled graph with following syntactic restrictions

- Source Vertex (subject): URI
- Edge label (predicate): URI
- Target Vertex (object): URI or scalar value

An edge, called a "triple", is the atomic unit

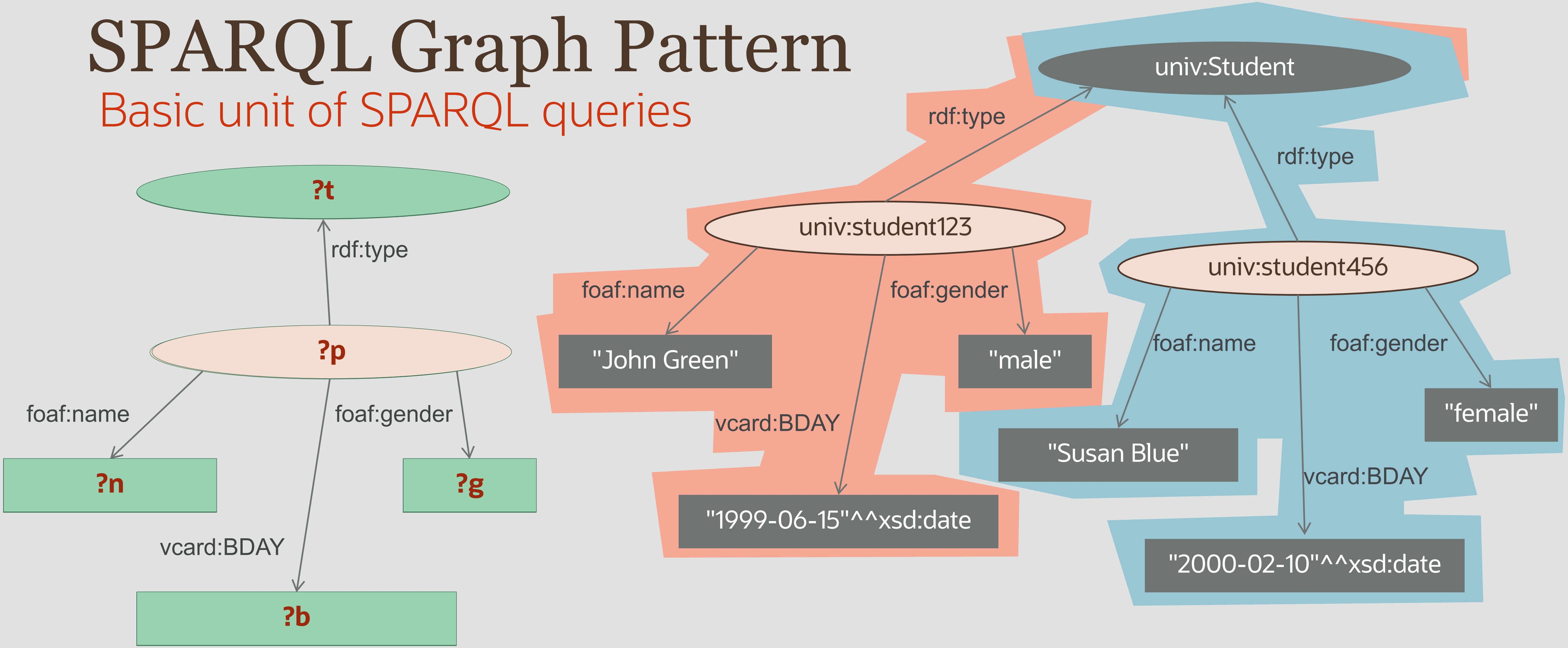
- Resource-Triple: < URI, URI, URI >
- Value-Triple: < URI, URI, value >



[1] "RDF does not place any formal restrictions on what resource the graph name may denote …"
SEE: https://www.w3.org/TR/rdf11-concepts/#section-dataset

SPARQL Graph Pattern

Basic unit of SPARQL queries

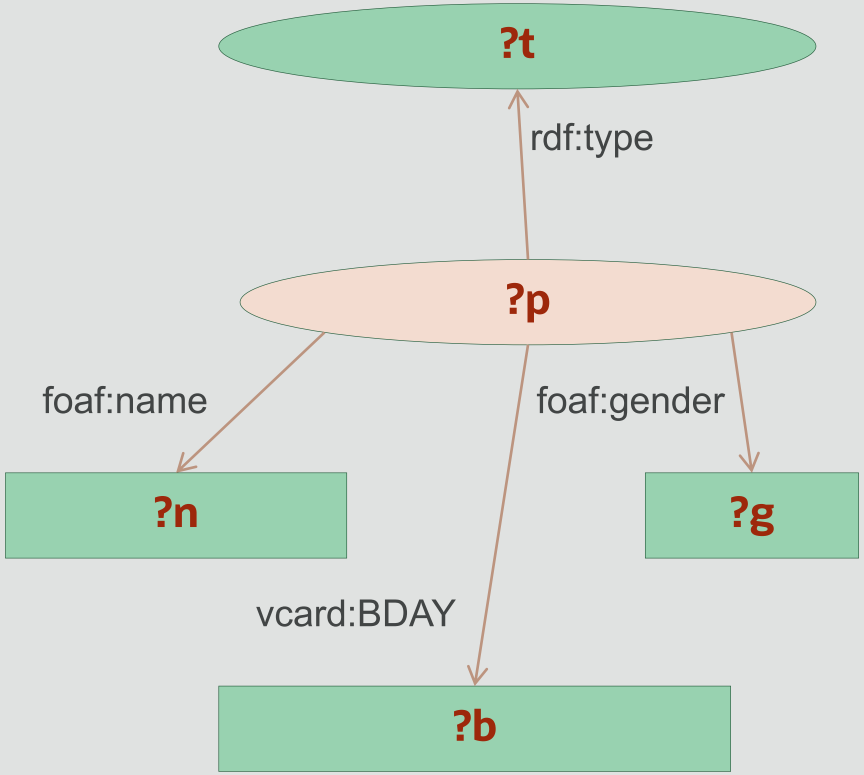


Result 1: {?t=univ:Student, ?p=univ:student123, ?n="John Green", ?g="male", ?b="1999-06-15"^^xsd:date}

Result 2: {?t=univ:Student, ?p=univ:student456, ?n="Susan Blue", ?g="female", ?b="2000-02-10"^^xsd:date}

SPARQL Graph Pattern

Basic unit of SPARQL queries



How do we express this with SPARQL?

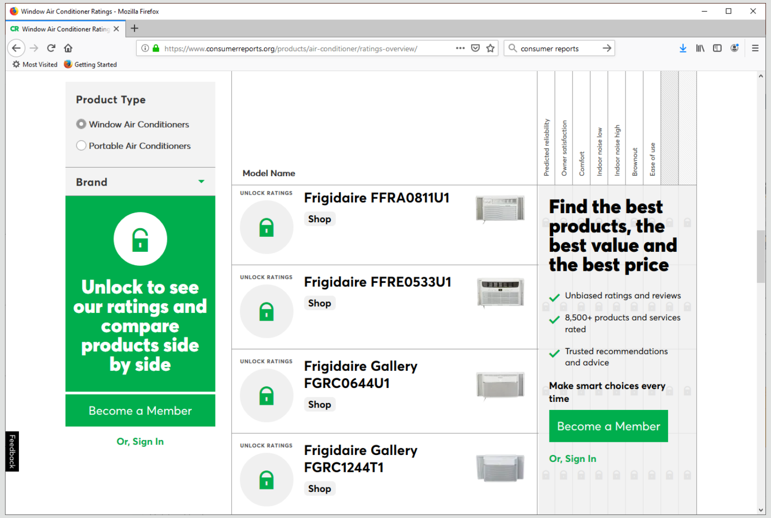
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?t ?n ?b ?g
WHERE
{ ?p rdf:type ?t ;
      foaf:name ?n ;
      vcard:BDAY ?b ;
      foaf:gender ?g }
```

Basic Graph Pattern (BGP)

Introduction to Linked Data Through an Example

CR Consumer Reports

amazon



Window Air Conditioner Ratings - Mozilla Firefox

https://www.consumerreports.org/products/air-conditioner/ratings-overview/

Product Type

● Window Air Conditioners

○ Portable Air Conditioners

Brand

Unlock to see our ratings and compare products side by side

Become a Member

Or, Sign In

Model Name	Predicted reliability	Owner satisfaction	Comfort	Indoor noise low	Indoor noise high	Brownout	Ease of use
UNLOCK RATINGS Frigidaire FFRA0811U1 Shop							
UNLOCK RATINGS Frigidaire FFRE0533U1 Shop							
UNLOCK RATINGS Frigidaire Gallery FGRC0644U1 Shop							
UNLOCK RATINGS Frigidaire Gallery FGRC1244T1 Shop							

Find the best products, the best value and the best price

✓ Unbiased ratings and reviews

✓ 8,500+ products and services rated

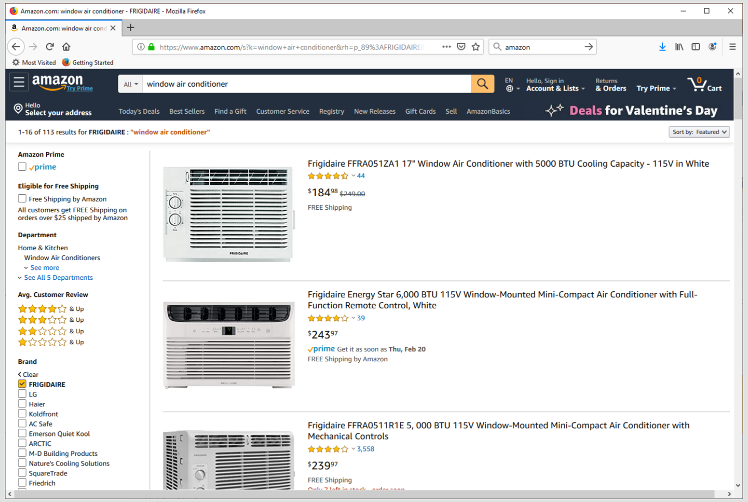
✓ Trusted recommendations and advice

Make smart choices every time

Become a Member

Or, Sign In

I need an air conditioner



Amazon.com: window air conditioner - FRIGIDAIRE - Mozilla Firefox

https://www.amazon.com/s?k=window+air+conditioner&rh=p_89%3AFRIGIDAIRE

window air conditioner

Hello, Sign in Account & Lists · Returns & Orders · Try Prime · Cart

Hello Select your address · Today's Deals · Best Sellers · Find a Gift · Customer Service · Registry · New Releases · Gift Cards · Sell · AmazonBasics · Deals for Valentine's Day

1-16 of 113 results for FRIGIDAIRE : "window air conditioner"

Sort by: Featured

Amazon Prime

□ prime

Eligible for Free Shipping

□ Free Shipping by Amazon

All customers get FREE Shipping on orders over $25 shipped by Amazon

Department

Home & Kitchen

Window Air Conditioners

See more

See All 5 Departments

Avg. Customer Review

★★★★☆ & Up

★★★☆☆ & Up

★★☆☆☆ & Up

★☆☆☆☆ & Up

Brand

Clear

☑ FRIGIDAIRE

□ LG

□ Haier

□ Koldfront

□ AC Safe

□ Emerson Quiet Kool

□ ARCTIC

□ M-D Building Products

□ Nature's Cooling Solutions

□ SquareTrade

□ Friedrich

Frigidaire FFRA051ZA1 17" Window Air Conditioner with 5000 BTU Cooling Capacity - 115V in White

★★★★☆ 44

$184.98 ~~$249.00~~

FREE Shipping

Frigidaire Energy Star 6,000 BTU 115V Window-Mounted Mini-Compact Air Conditioner with Full-Function Remote Control, White

★★★★☆ 39

$243.97

prime Get it as soon as Thu, Feb 20

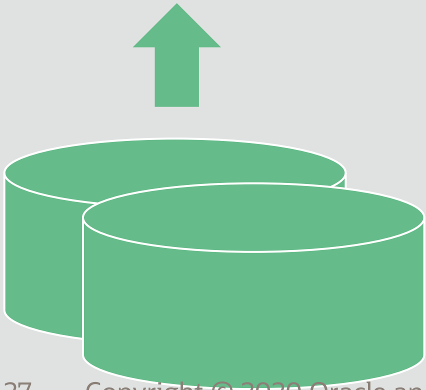
FREE Shipping by Amazon

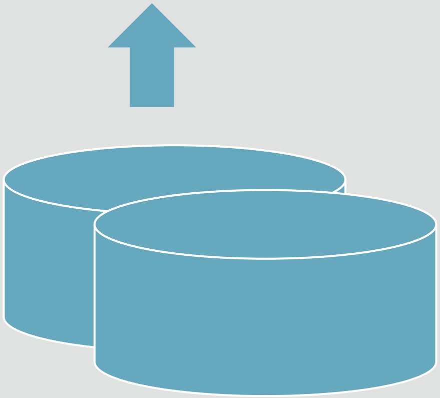
Frigidaire FFRA0511R1E 5, 000 BTU 115V Window-Mounted Mini-Compact Air Conditioner with Mechanical Controls

★★★★☆ 3,558

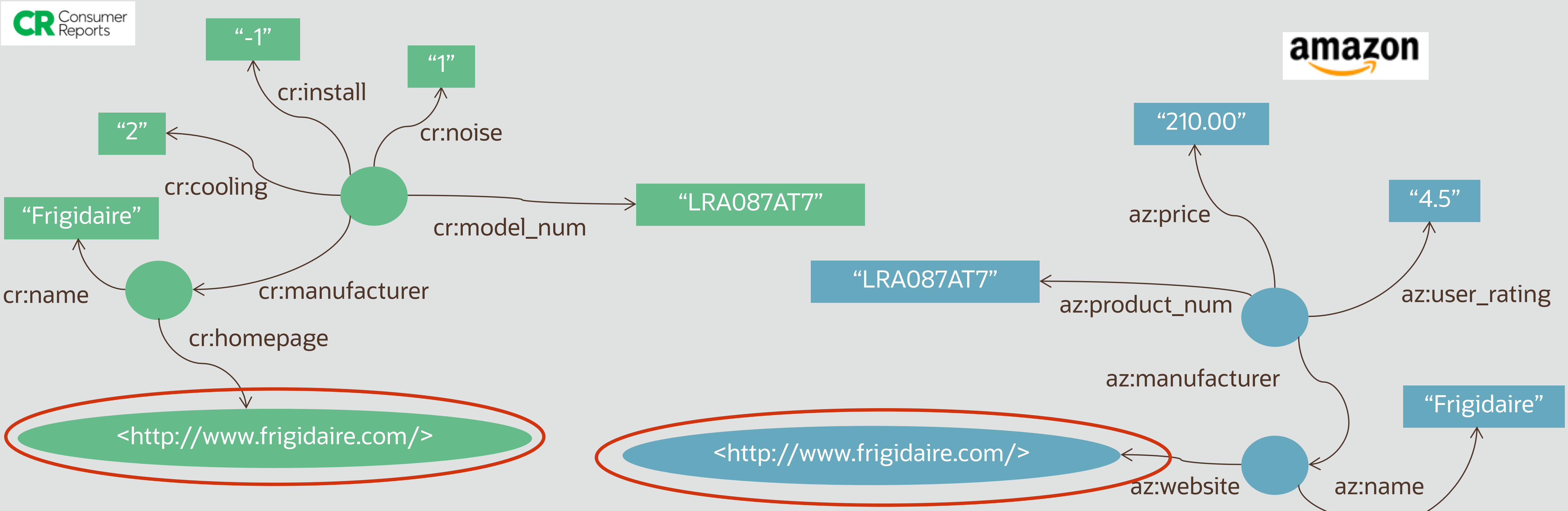
$239.97

FREE Shipping



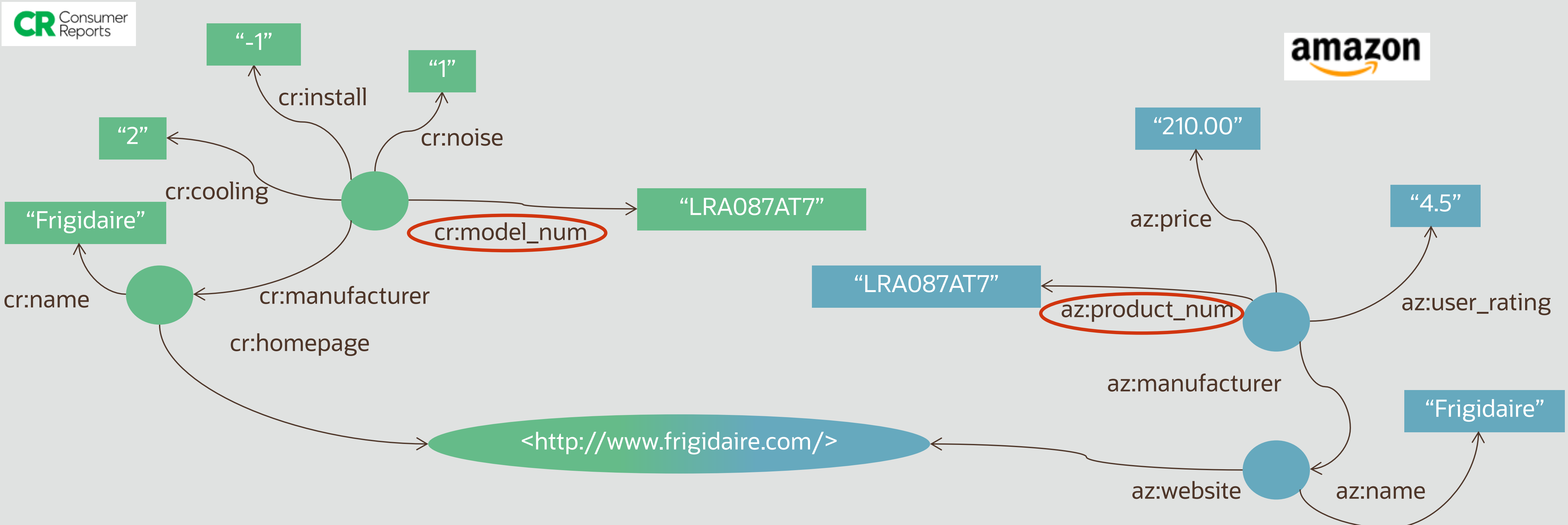


Introduction to Linked Data Through an Example



Same URI

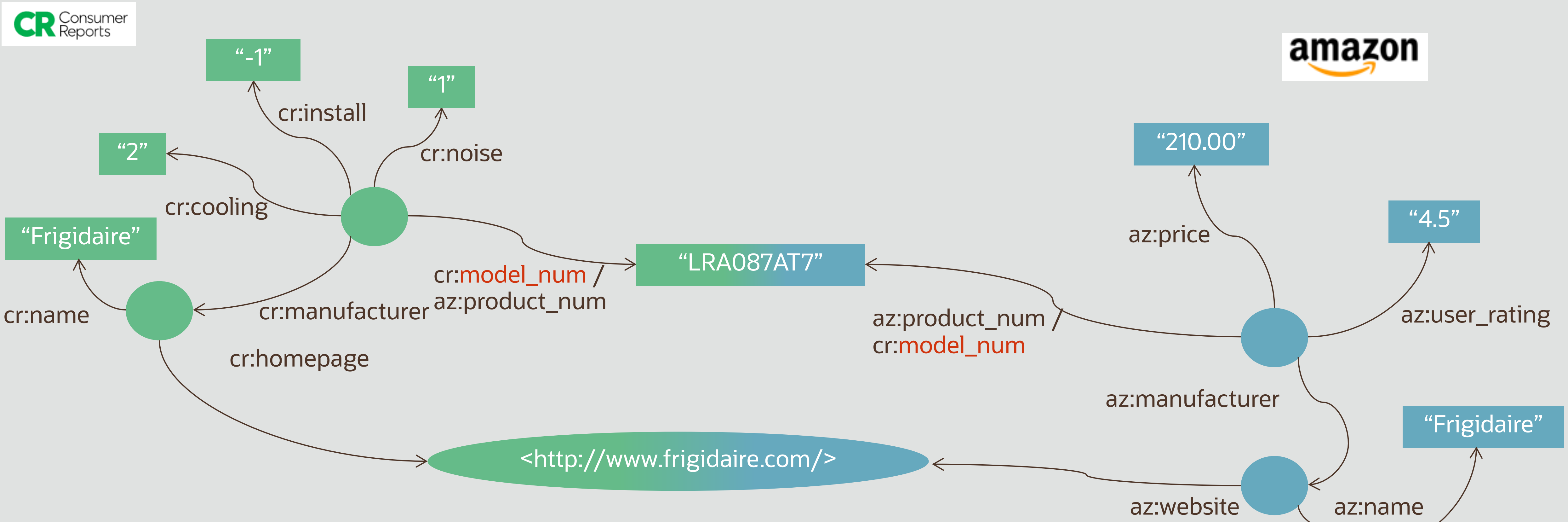
Introduction to Linked Data Through an Example



Suspect that cr:model_num is the same as az:product_num

cr:model_num **owl:equivalentProperty** az:product_num

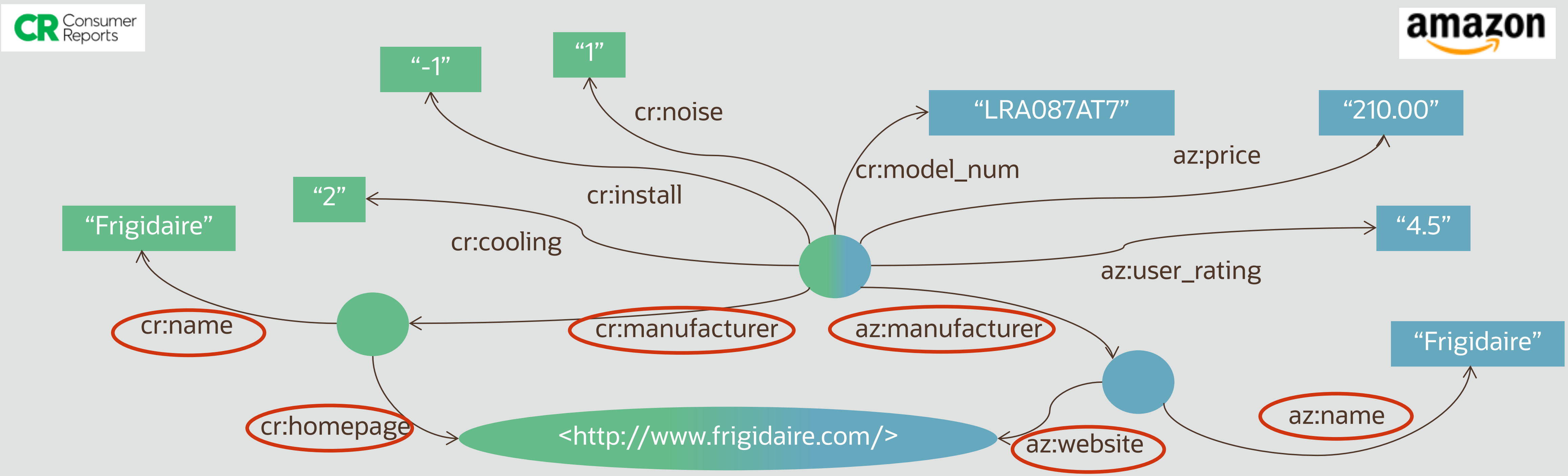
Introduction to Linked Data Through an Example



We suspect that model number is unique

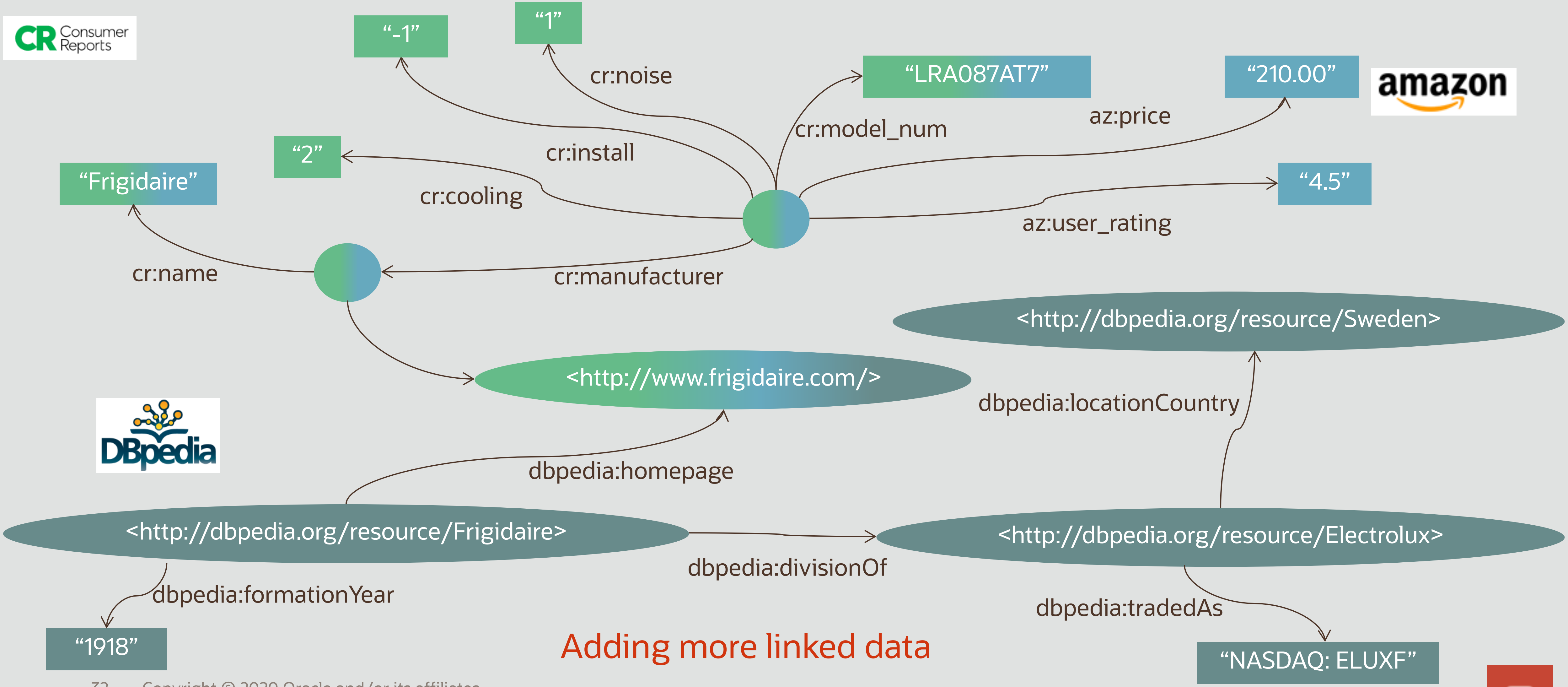
cr:model_num rdf:type **owl:inverseFunctionalProperty**

Introduction to Linked Data Through an Example



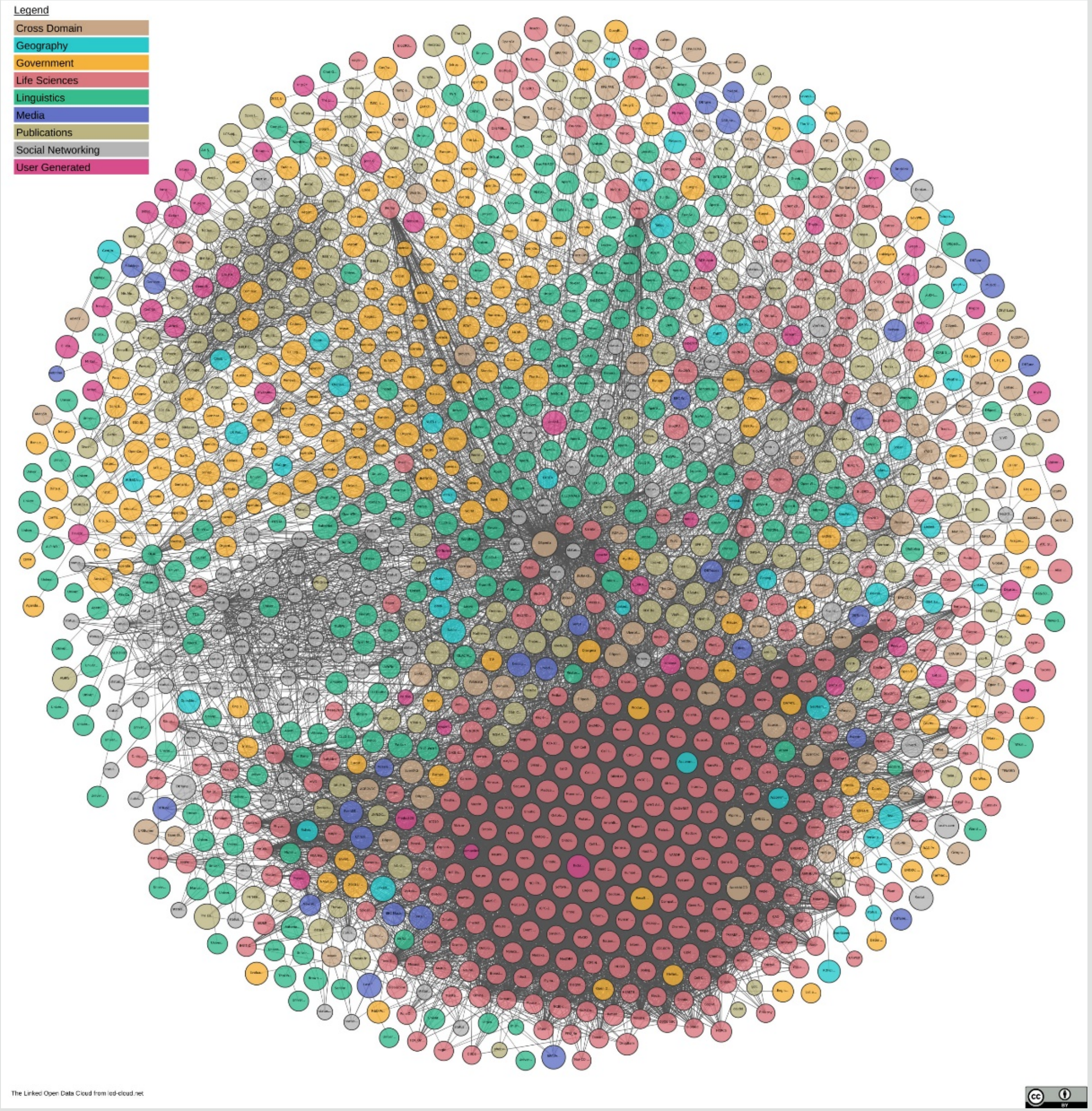
cr:manufacturer **owl:equivalentProperty** az:manufacturer
cr:homepage **owl:equivalentProperty** az:website
cr:homepage rdf:type **owl:inverseFunctionalProperty**
cr:name **owl:equivalentProperty** az:name

Introduction to Linked Data Through an Example



Adding more linked data

A Big RDF Graph Linking Many Data Sources



https://lod-cloud.net/ 1,200+ linked datasets

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Resources for Getting Started

- VM image: : https://www.oracle.com/database/technologies/databaseappdev-vm.html

- Oracle Database Docker

  Single instance database from

  https://github.com/oracle/docker-images/tree/master/OracleDatabase

- Oracle Cloud

  Use **Oracle Database Cloud Service** with $300 free credits

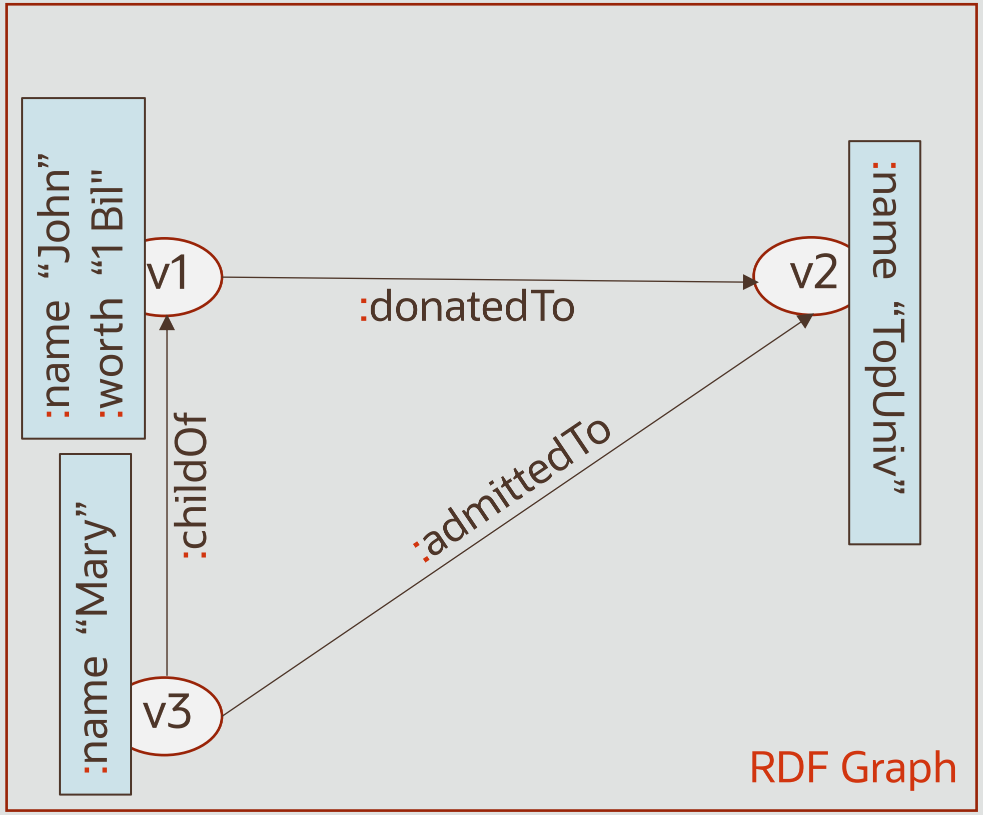
  On the roadmap: RDF Graph support in 'Always Free Tier'

Implementing in RDF: Vertex, Edge, Vertex-Property

John, whose net worth is $1 billion, donated to Top University.
Mary, a child of John, got admitted to Top University.

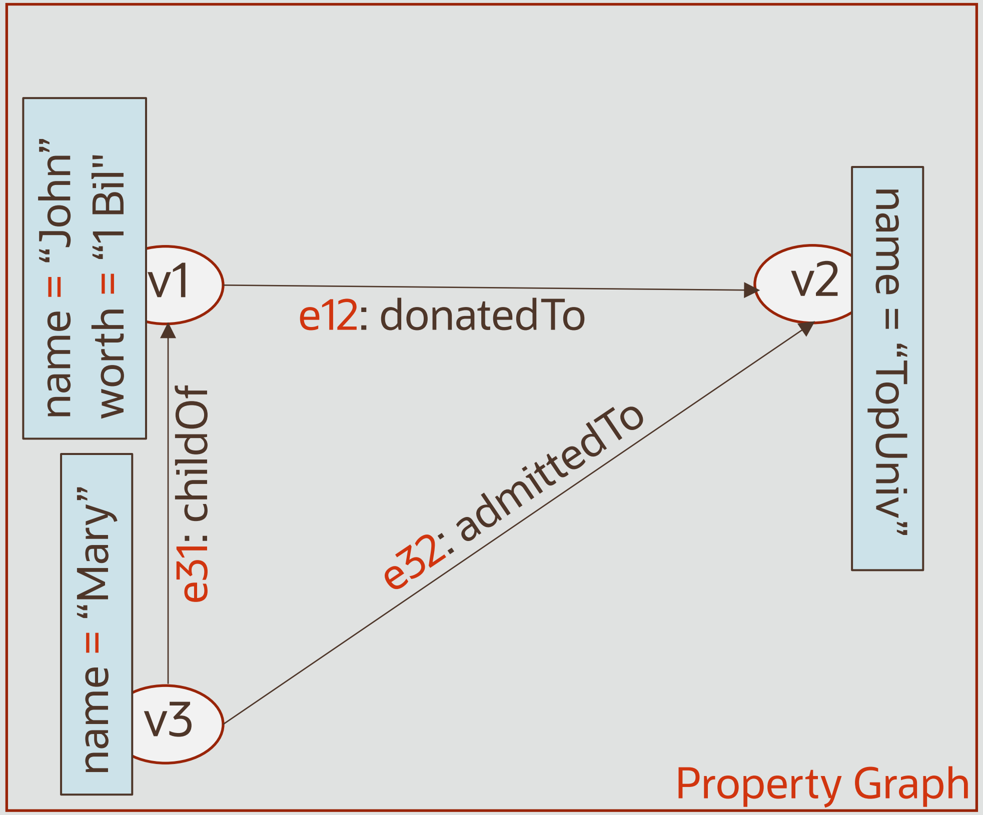
```
BEGIN
 sem_apis.update_model(‘rdf_demo_graph',
  'PREFIX : <http://demo/>
   INSERT DATA {
     :v1  :name          “John” ;
          :worth          “1 Bil” ;
          :donatedTo   :v2 .
     :v2  :name          “TopUniv” .
     :v3  :name          “Mary” ;
           :admittedTo :v2 ;
           :childOf        :v1 .
   } ‘, network_owner=>’..’,network_name=>’..’);
END;
/
```

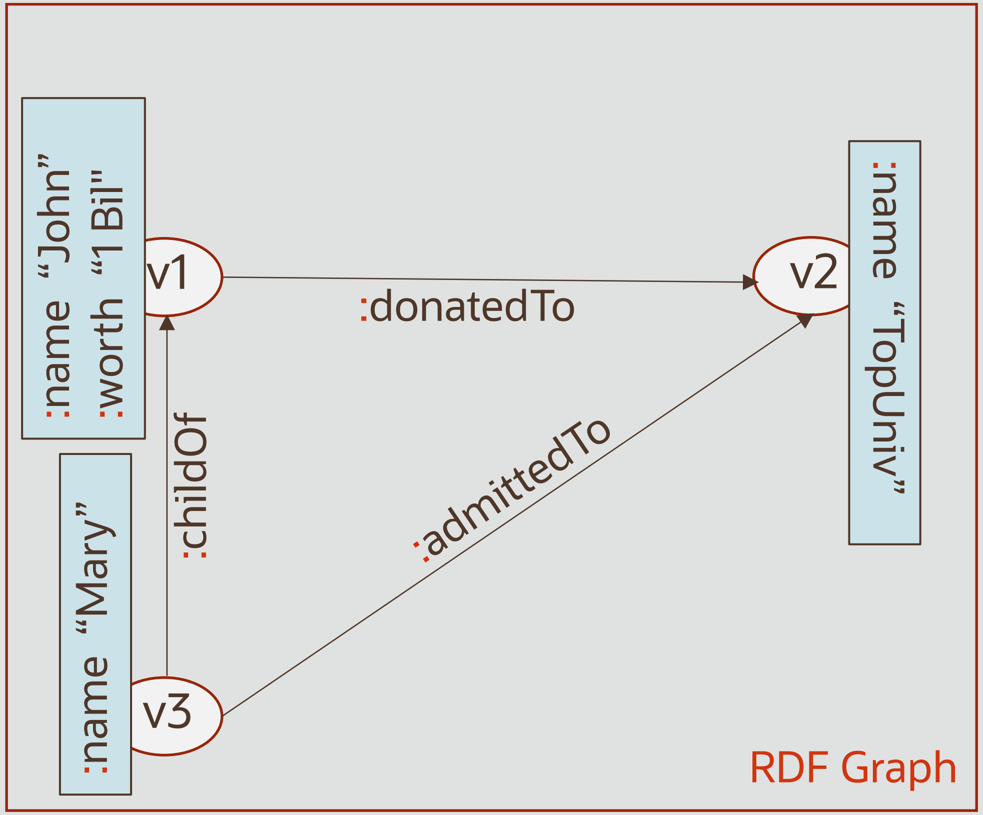
SPARQL Update



Graphs in PG and RDF: Vertex, Edge, Vertex-Property

John, whose net worth is $1 billion, donated to Top University.
Mary, a child of John, got admitted to Top University.



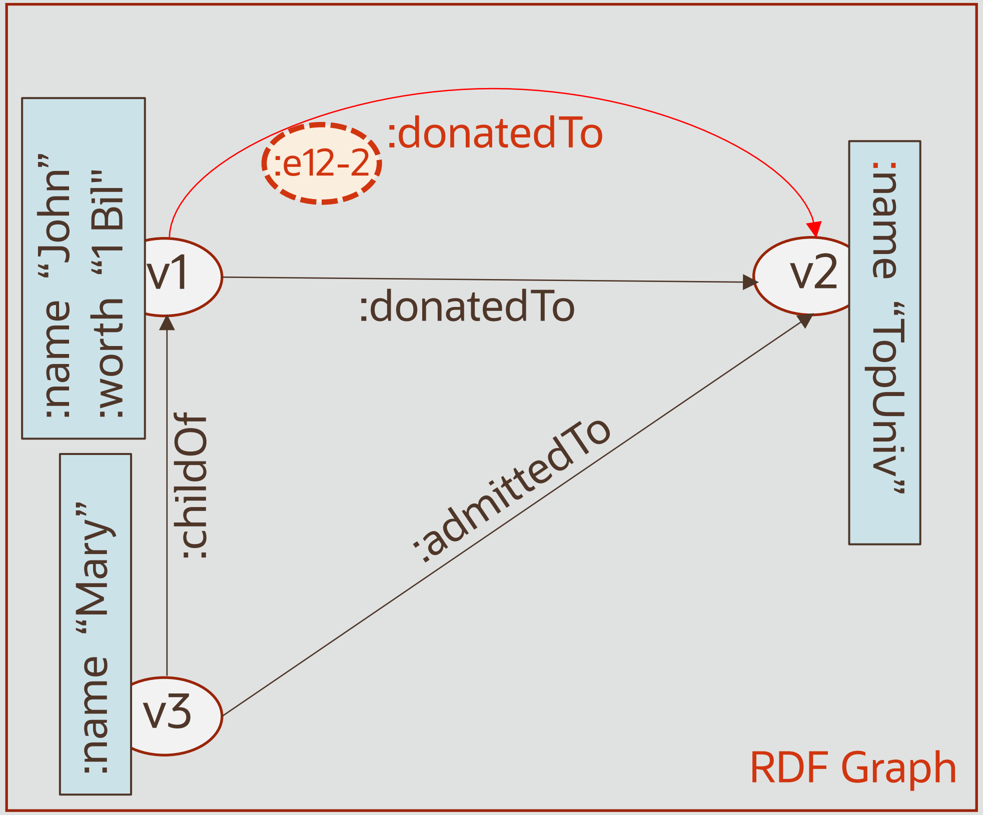


Implementing in RDF: Duplicate Edge

John … donated twice to Top University. …

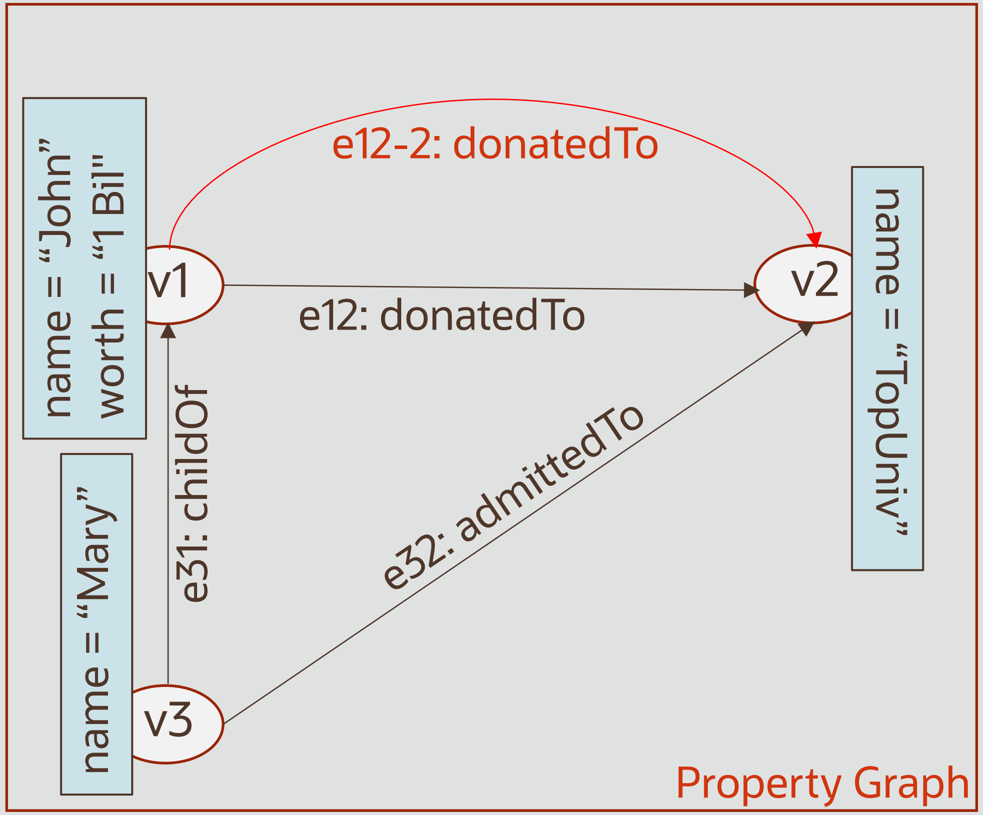
```
BEGIN
 sem_apis.update_model(‘rdf_demo_graph',
  'PREFIX : <http://demo/>
   INSERT DATA {
     graph :e12-2 { :v1 :donatedTo :v2 }
   } ');
END;
/
```

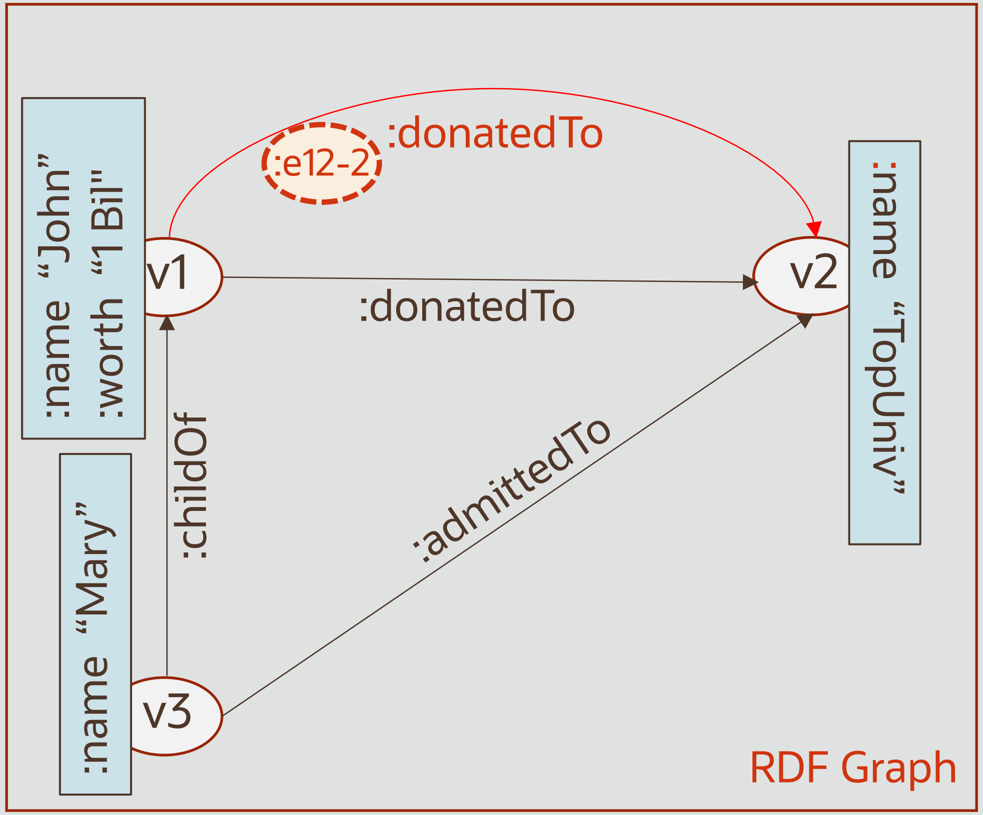
SPARQL Update



Graphs in PG and RDF: Duplicate Edge

John … donated twice to Top University. …



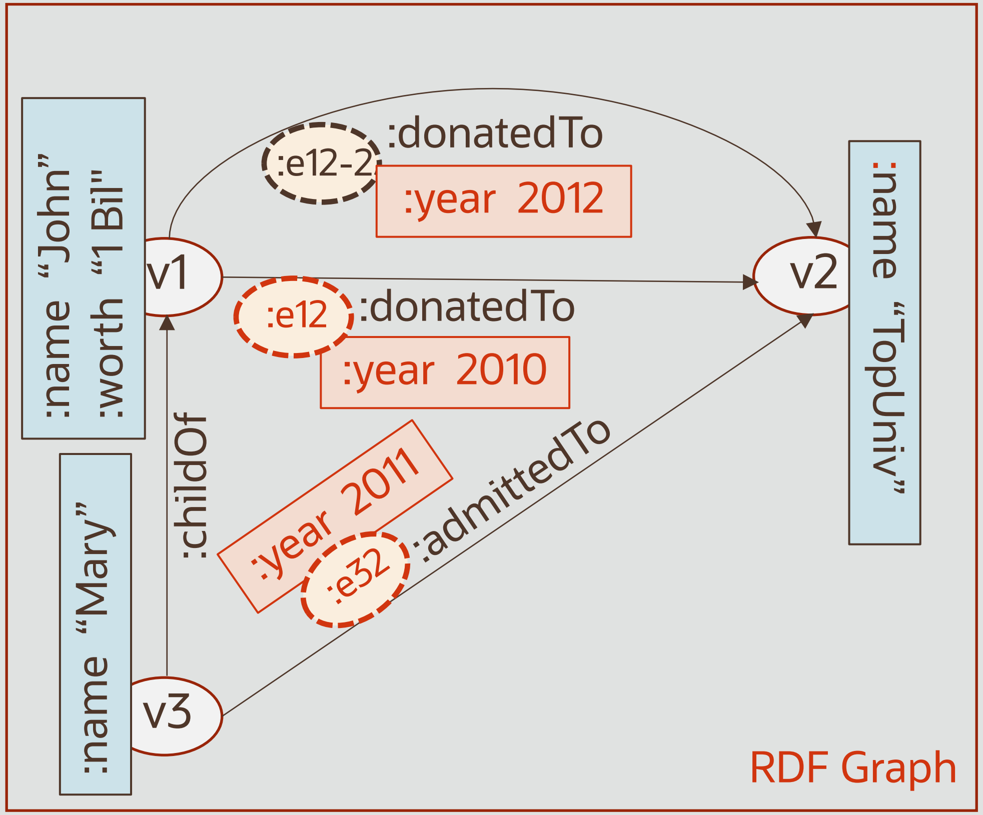


Implementing in RDF: Edge-Property

John … donated twice to Top University, in the years 2010 and 2012, respectively.
Mary … got admitted to Top University in 2011.

```
BEGIN
  sem_apis.update_model('rdf_demo_graph',
   'PREFIX : <http://demo/>
    DELETE DATA {
      :v1 :donatedTo  :v2 . # deletes triple ONLY
      :v3 :admittedTo :v2 .
    } ;
    INSERT DATA {
      graph :e12   { :v1 :donatedTo  :v2 }
      graph :e32   { :v3 :admittedTo :v2 }
      :e12      :year 2010 .
      :e12-2  :year 2012 .
      :e32     :year 2011 .
    } ');
END;
```

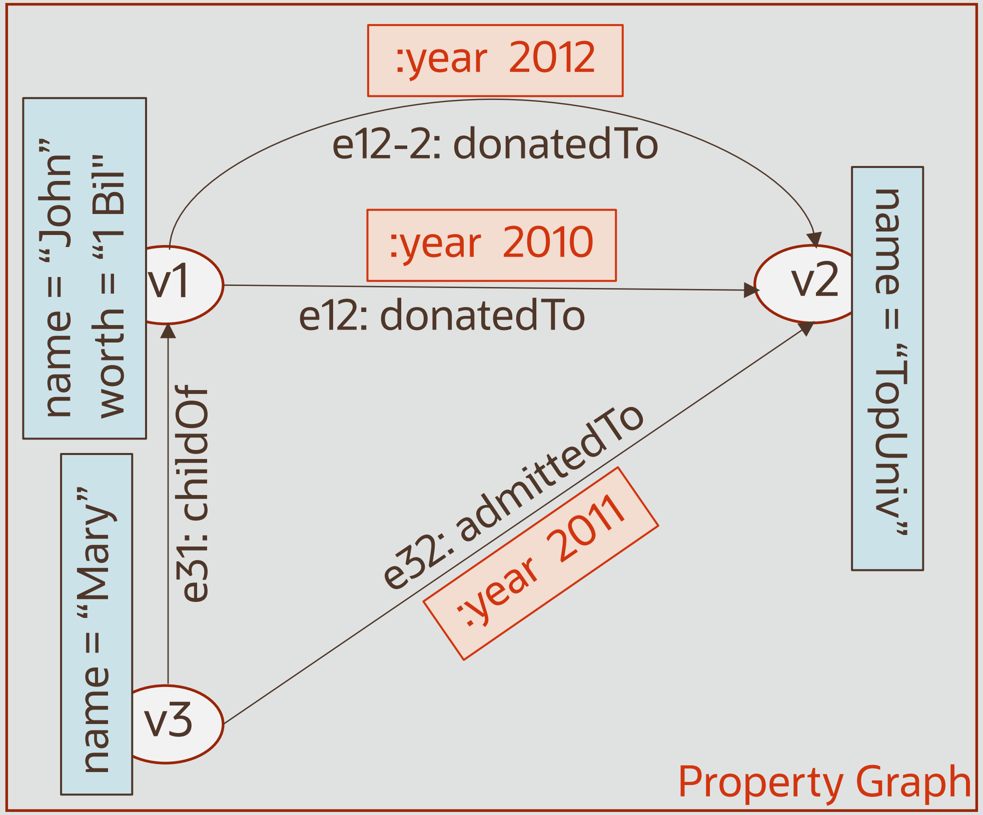
SPARQL Update

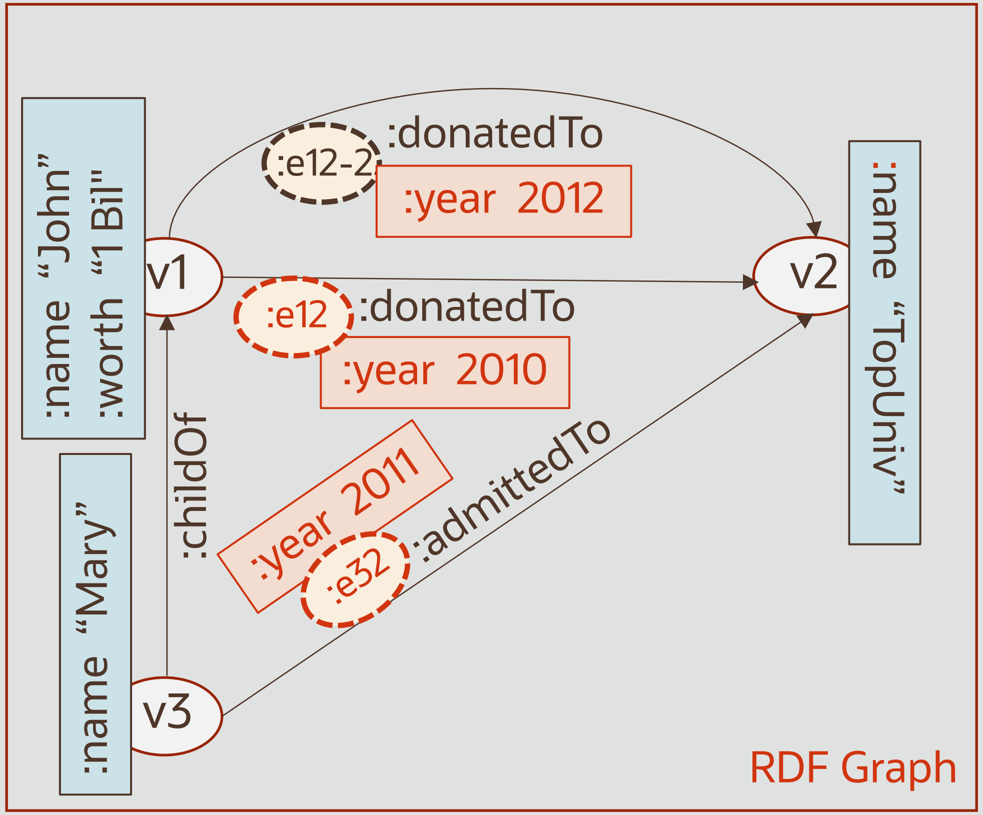


RDF Graph

Graphs in PG and RDF: Edge-Property

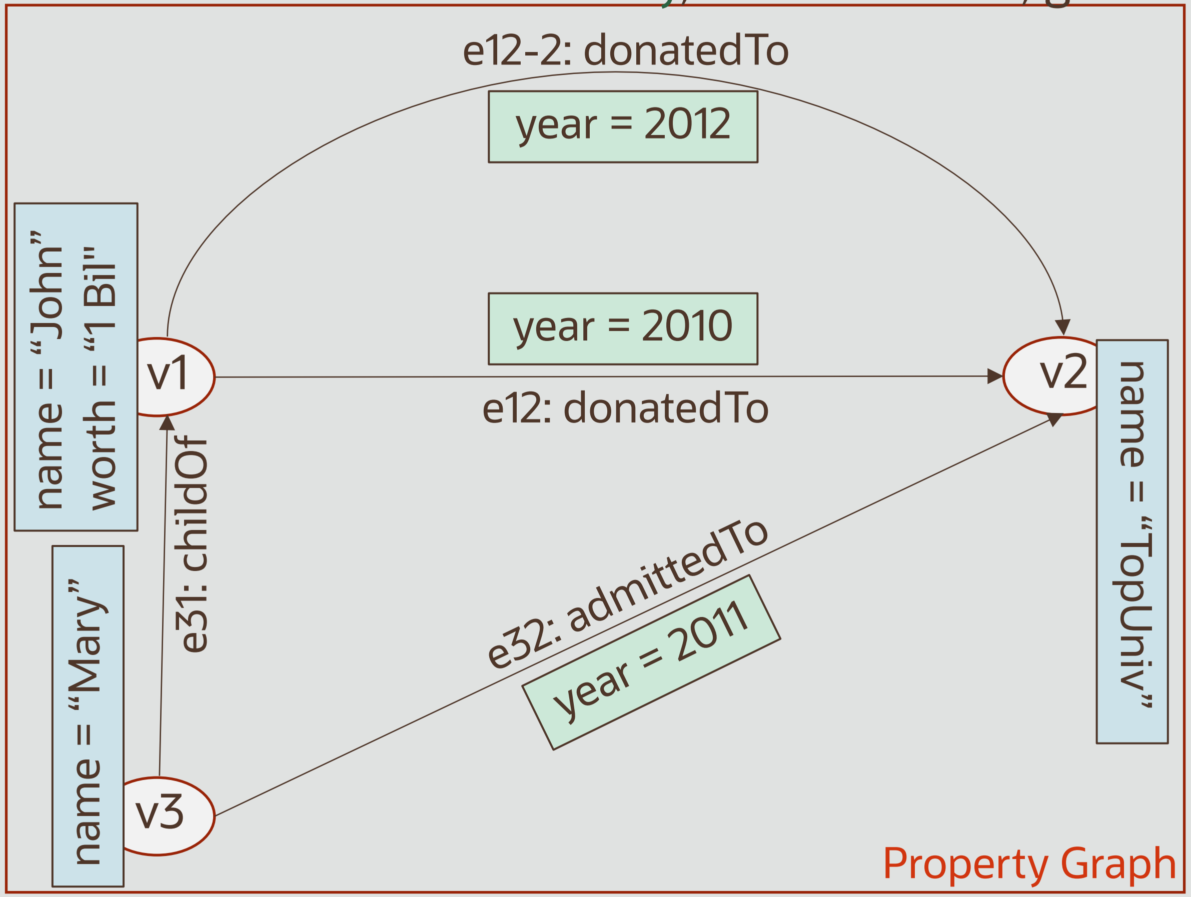
John … donated twice to Top University, in the years 2010 and 2012, respectively.
Mary … got admitted to Top University in 2011.

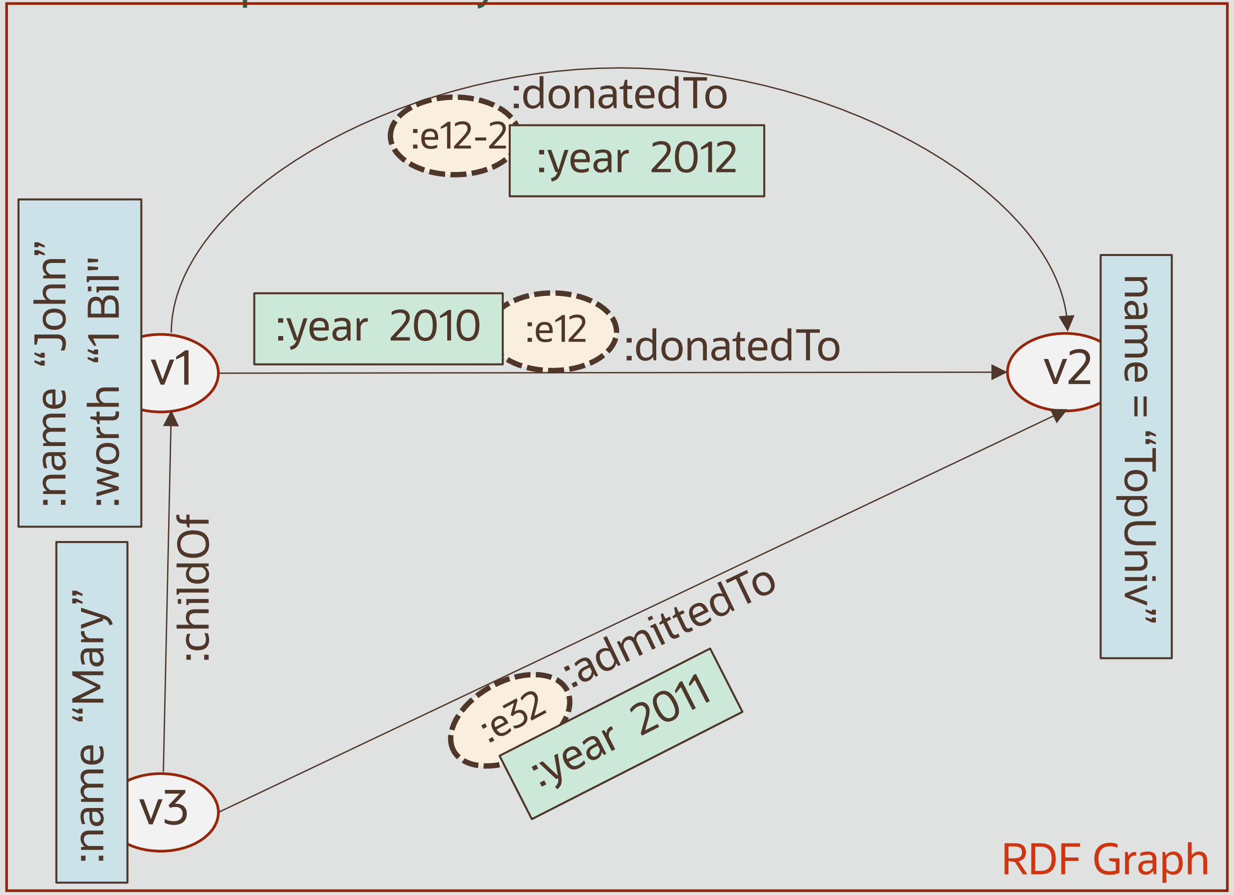




RDF via PG-lens: The Graphs at this point. Vertex, Edge, Vertex- and Edge-Properties

John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.
Mary, a child of John, got admitted to Top University in 2011.

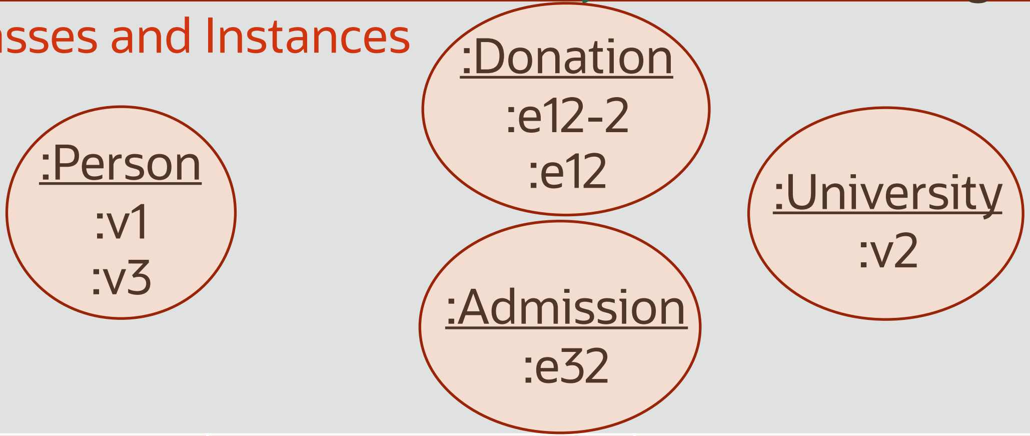




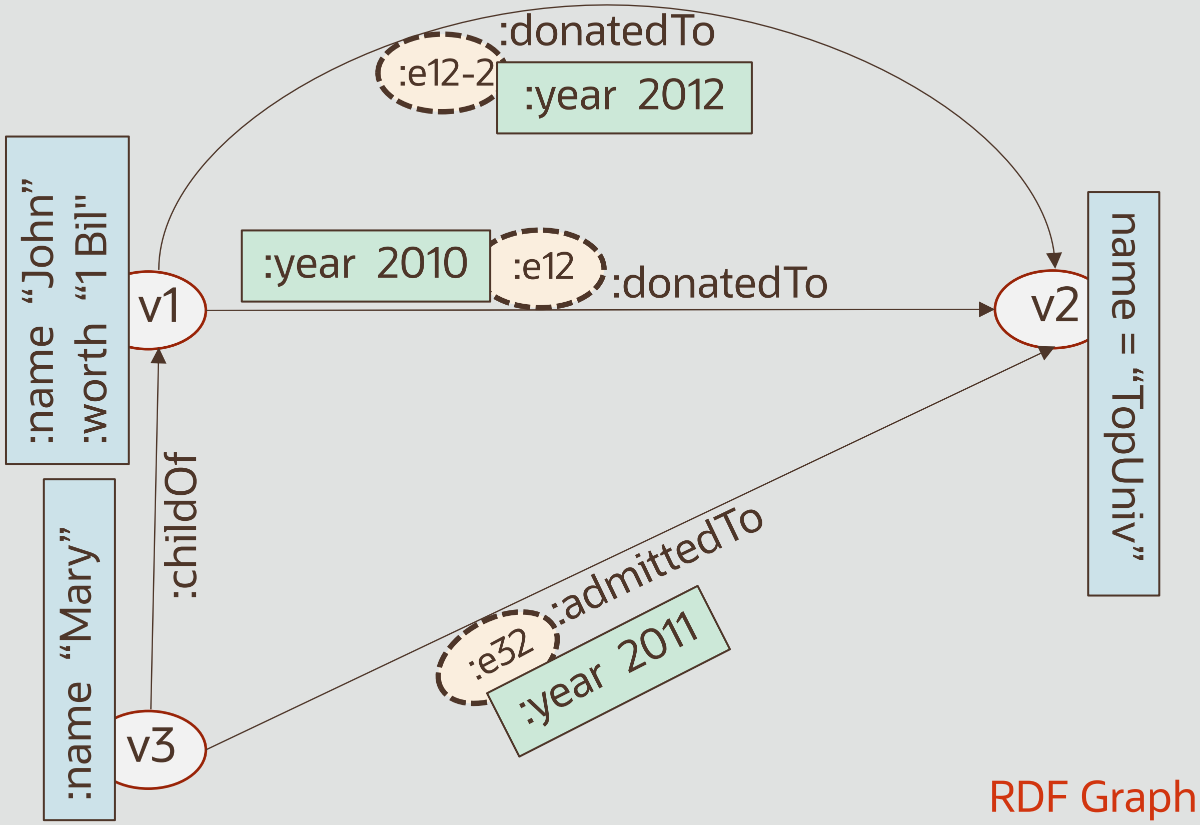
Implementing in RDF:
Schema for Resulting RDF Graph

John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.
Mary, a child of John, got admitted to Top University in 2011.

Classes and Instances



relation	domain	range
:admittedTo	:Person	:University
:childOf	:Person	:Person
:donatedTo	:Person	:University
:name	:Person, :University	xsd:string
:worth	:Person	xsd:string
:year	:Admission, :Donation	xsd:decimal



RDF Graph

Implementing in RDF: SPARQL Query Vertex, Edge, Vertex- and Edge-Properties

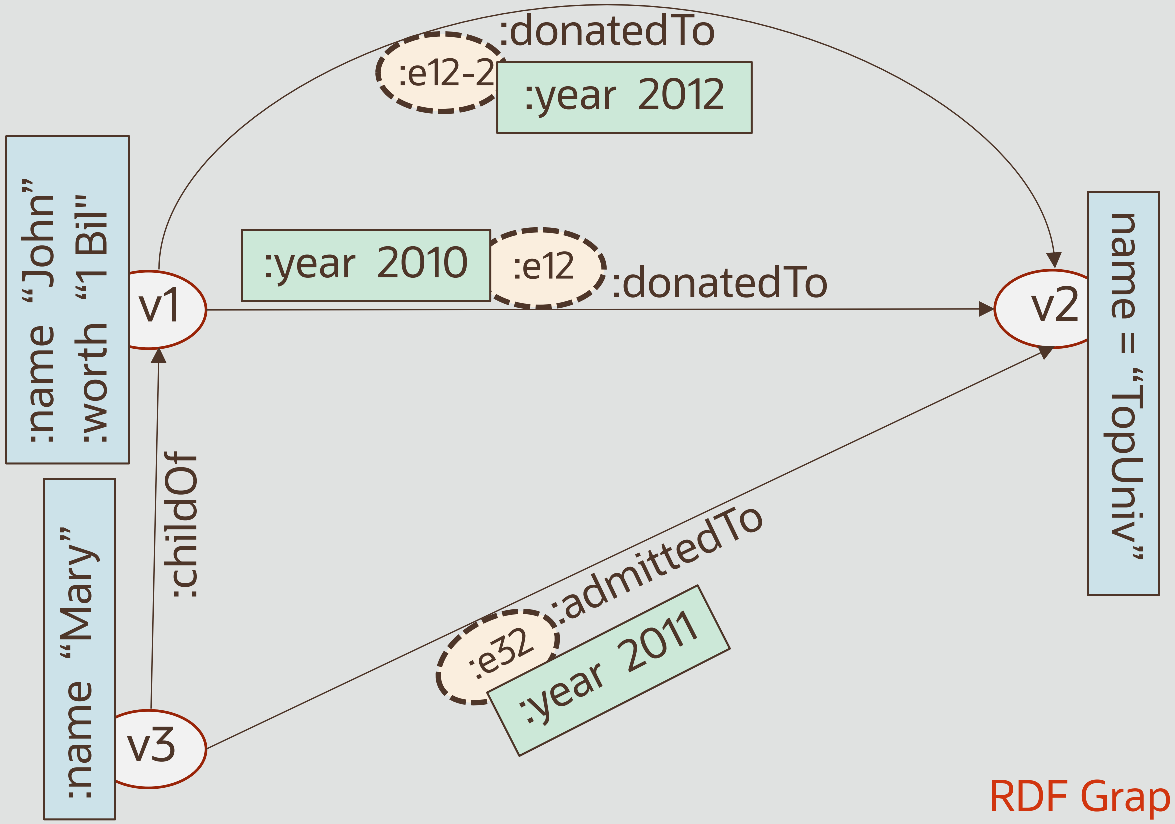
John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.
Mary, a child of John, got admitted to Top University in 2011.

Find names of parent, university, and child where parent donated to the university during a year and his/her child got admitted to the university in the following year.

```
SELECT ?paName ?univName ?chName
WHERE {
  ?child      :childOf    ?parent .
  #
  graph ?donEdge { ?parent    :donatedTo    ?univ }
  ?donEdge    :year         ?donYear .
  #
  graph ?admEdge { ?child      :admittedTo   ?univ }
  ?admEdge    :year         ?admYear .
  #
  FILTER ( ?admYear = ?donYear + 1 )
  ?child      :name         ?chName .
  ?parent     :name         ?paName .
  ?univ       :name         ?univName }
```

triple name is specified as graph name.

SPARQL Query



RDF Graph

Implementing in RDF: SPARQL# Query Vertex, Edge, Vertex- and Edge-Properties

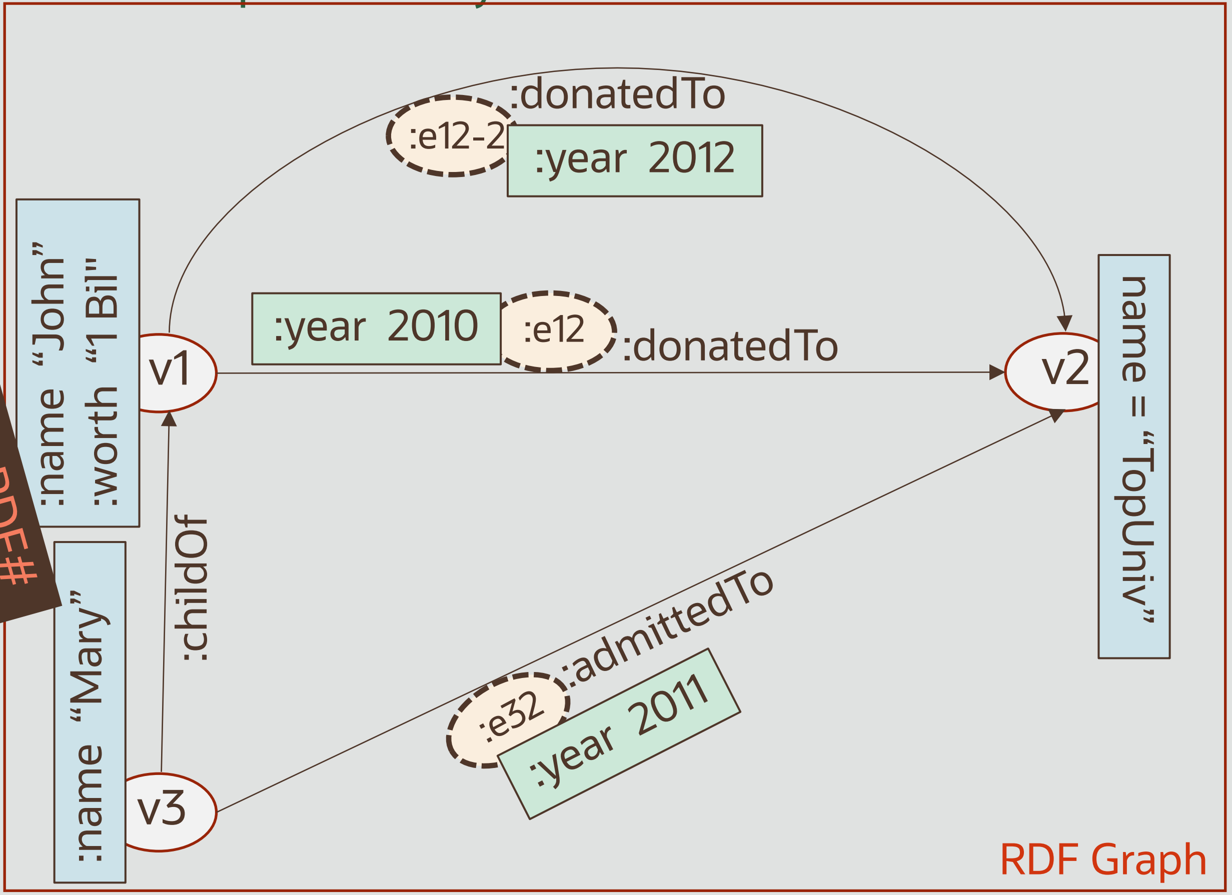
John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.
Mary, a child of John, got admitted to Top University in 2011.

Find names of parent, university, and child where parent donated to the university during a year and his/her child got admitted to the university in the following year.

```
SELECT ?paName ?univName ?chName
WHERE {
  ?child      :childOf     ?parent .
  #
  ?parent    [?donEdge]:donatedTo    ?univ .
  ?donEdge    :year        ?donYear .
  #
  ?child     [?admEdge]:admittedTo   ?univ .
  ?admEdge    :year        ?admYear .
  #
  FILTER ( ?admYear = ?donYear + 1 )
  ?child     :name       ?chName .
  ?parent    :name       ?paName .
  ?univ      :name       ?univName }
```

SPARQL Query in RDF#

SPARQL using RDF# syntax: triple name is piggybacked on the predicate, not as graph name.

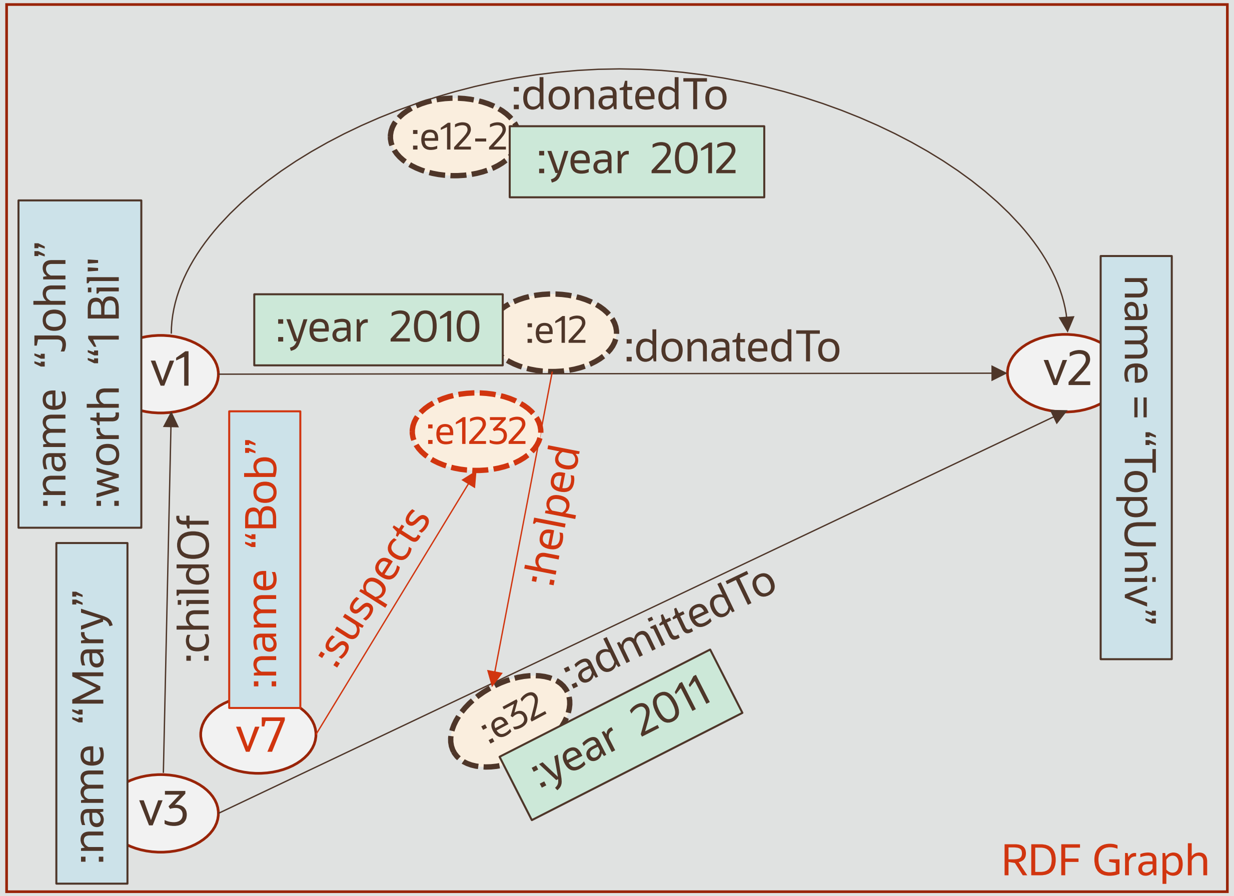


Implementing in RDF: Edges as Endpoints of Another Edge

… Bob suspects that John’s 2010 donation helped Mary’s admission.

```
BEGIN
  sem_apis.update_model('rdf_demo_graph',
   'PREFIX : <http://demo/>
    INSERT DATA {
      graph :e1232 { :e12 :helped :e32 }
      :v7      :name       “Bob” ;
               :suspects   :e1232
    } ');
END;
/
```

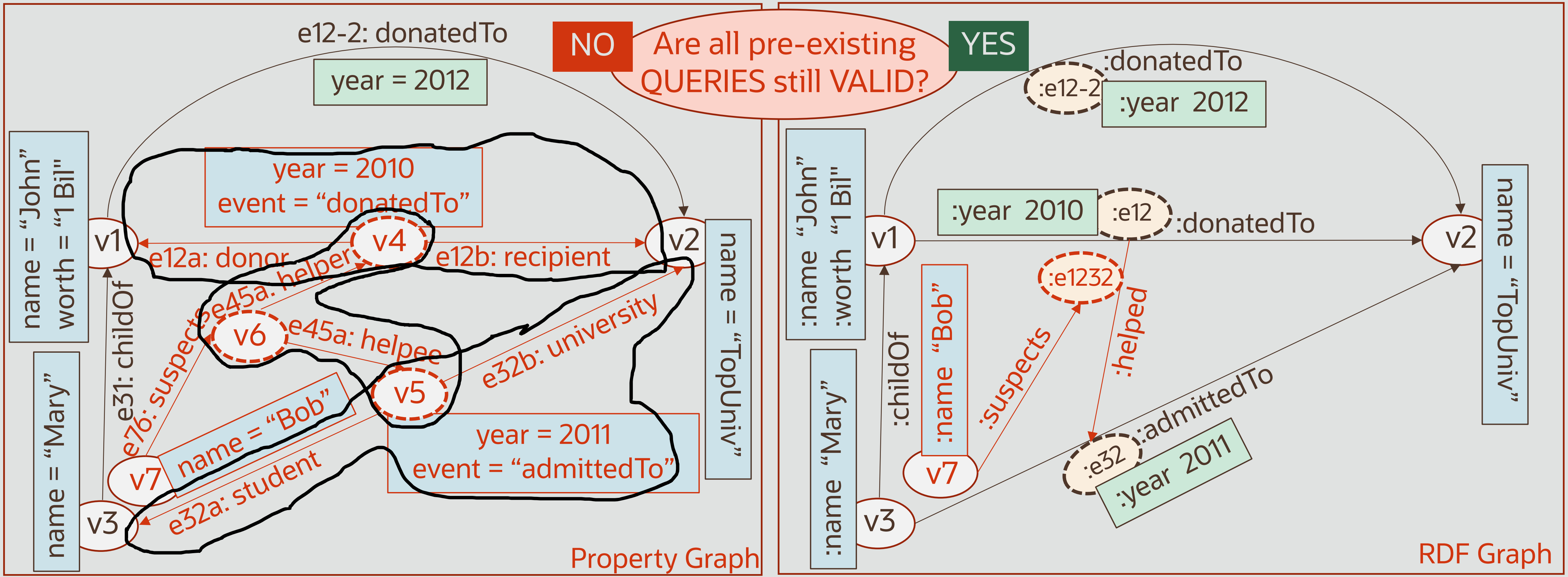
SPARQL Update



RDF Graph

Graphs in PG & RDF: Backward-Compatible?
Edges as Endpoints of Another Edge

… Bob suspects that John's 2010 donation helped Mary's admission.



NO

Are all pre-existing QUERIES still VALID?

YES

Implementing in RDF:
Schema for Resulting RDF Graph

John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively. Mary, a child of John, got admitted to Top University in 2011. Bob suspects that the 2010 donation helped the 2011 admission.

How was the schema affected?

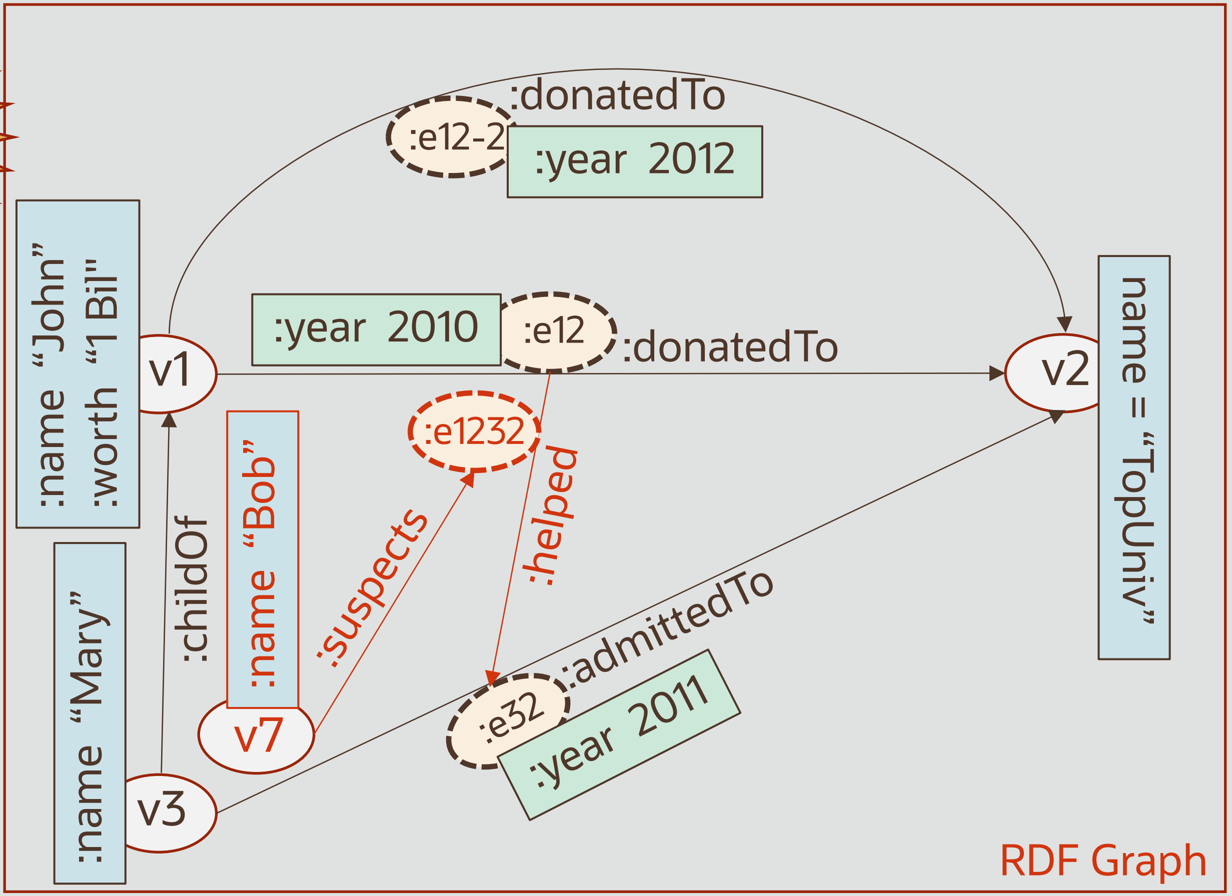
- one new class
- Two new relations

:Helping
:e1232

Nothing got dropped!

relation	domain	range
:admittedTo	:Person	:University
:childOf	:Person	:Person
:donatedTo	:Person	:University
:helped	**:Donation**	**:Admission**
:suspects	**:Person**	**:Helping**
:name	:Person, :University	xsd:string
:worth	:Person	xsd:string
:year	:Admission, :Donation	xsd:decimal

Schema for RDF Data



RDF Graph

Implementing in RDF:
Resulting RDF Graph, SPARQL query

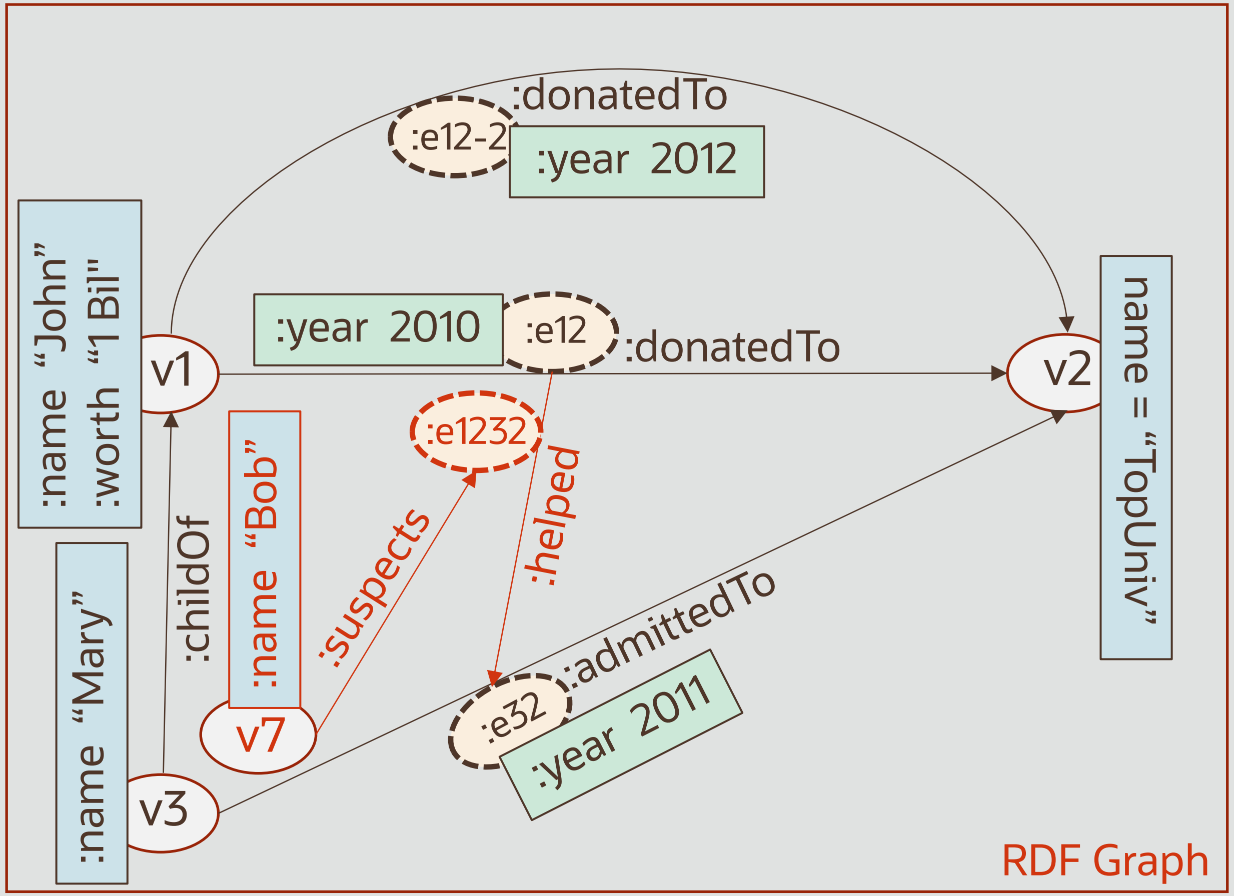
John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively. Mary, a child of John, got admitted to Top University in 2011. Bob suspects that the 2010 donation helped the 2011 admission.

Find names of parent, university, and child where parent donated to the university during a year and his/her child got admitted to the university in the following year.

```
SELECT ?paName ?univName ?chName
WHERE {
  ?child     :childOf    ?parent .
  #
  graph ?donEdge { ?parent    :donatedTo    ?univ }
  ?donEdge    :year         ?donYear .
  #
  graph ?admEdge { ?child     :admittedTo   ?univ }
  ?admEdge   :year         ?admYear .
  #
  FILTER ( ?admYear = ?donYear + 1 )
  ?child     :name        ?chName .
  ?parent  :name        ?paName .
  ?univ      :name        ?univName }
```

All pre-existing queries remain valid.

SPARQL Query



RDF Graph

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

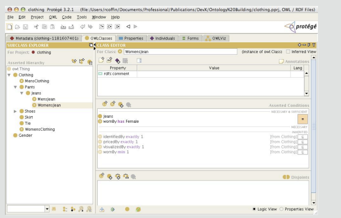
- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Demo Environment for Tutorial

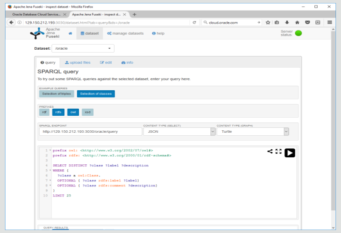
- Using a freely-available Virtual Machine image with Oracle Database 19.3
  - Other Software
    - Oracle Graph Server and Client 20.1
    - Oracle Support for Apache Jena 3.1.0
    - Java 11
- Using Linked Movie Data Base RDF Data
  - From a University of Toronto project
- Detailed setup information is available in a recent Oracle blog post: https://blogs.oracle.com/oraclespatial/kgc-2020-tutorial3a-modeling-evolving-data-in-graphs-while-preserving-backward-compatibility

Oracle Spatial and Graph 19c – RDF Knowledge Graph Architecture

Protégé Plugin



Fuseki Endpoint



SPARQL query

To try out some SPARQL queries against the selected dataset, enter your query here.

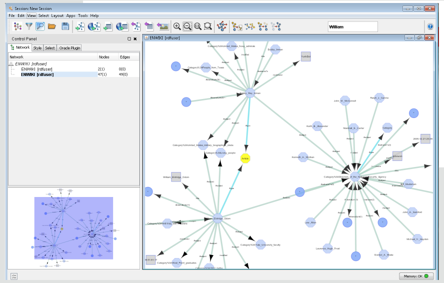
SPARQL ENDPOINT: http://129.150.212.193:3030/oracle/query

CONTENT TYPE (SELECT): JSON

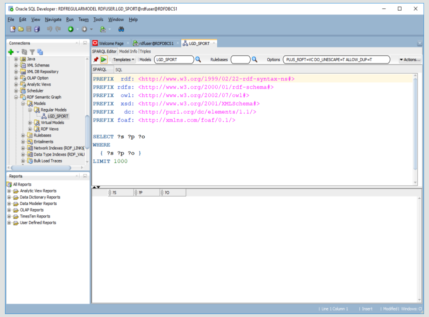
CONTENT TYPE (GRAPH): Turtle

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?class ?label ?description
WHERE {
  ?class a owl:Class.
  OPTIONAL { ?class rdfs:label ?label}
  OPTIONAL { ?class rdfs:comment ?description}
}
LIMIT 25
```

Cytoscape Plugin



SQL Developer RDF Support

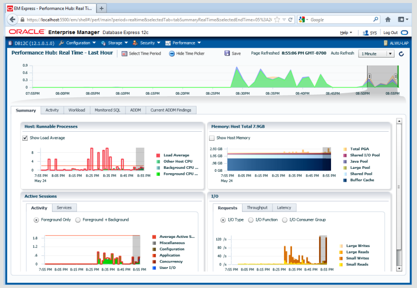


Models: LGD_SPORT

Options: PLUS_RDFT=VC DO_UNESCAPE=T ALLOW_DUP=T

```
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX   dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?s ?p ?o
WHERE
  { ?s ?p ?o }
LIMIT 1000
```

Enterprise Manager and Other DB Tools



Support for Apache Jena (Java API)

ORACLE®
DATABASE

SQL and PL/SQL API

RDF Bulk Loader

Forward-chaining OWL Reasoner

SPARQL-to-SQL Query Translator

SPARQL Update Processor

Generic Relational Schema for Storing RDF Data

RDF Views of Relational Data

Agenda

1 Graph Query Languages

2 Essentials for SPARQL Query & Update

3 Named Graphs for Edge Properties

4 Comparison with PG Query Languages

5 Graph Analytics with RDF Data

Agenda

1 Graph Query Languages

2 Essentials for SPARQL Query & Update

3 Named Graphs for Edge Properties

4 Comparison with PG Query Languages

5 Graph Analytics with RDF Data

Graph Query Languages

RDF Graph

SPARQL 1.1

W3C Semantic Web

SPARQL 1.2
SPARQL*

Property Graph

PGQL

ORACLE

G-CORE

LDBC The graph & RDF benchmark reference

Cypher

neo4j

SQL/PGQ
GQL

ISO International Organization for Standardization

GSQL

TigerGraph

Gremlin

Gremlin $G = (V, E)$

Agenda

1 Graph Query Languages

2 Essentials for SPARQL Query & Update

3 Named Graphs for Edge Properties

4 Comparison with PG Query Languages

5 Graph Analytics with RDF Data

Agenda

1 What is SPARQL

2 SPARQL 1.1 Query Features by Example

3 Graph Patterns

4 Property Paths

5 Named Graphs

6 Federated Queries

What is SPARQL?

- SPARQL Protocol and RDF Query Language
  - W3C standard for querying and manipulating RDF content
  - Queries/updates and corresponding results are communicated via HTTP with a *SPARQL endpoint*
  - A SPARQL endpoint implements the SPARQL protocol and serves RDF data from a *RDF triplestore* or *RDF view*

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

A comprehensive query language for RDF

Many useful constructs: optional patterns, aggregates, subqueries, negation, property paths, extensive function library, etc.

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

A comprehensive language for manipulating RDF graphs

Allows you to create, update and remove RDF graphs

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

Defines a protocol for sending queries or updates to SPARQL endpoint and returning the results via HTTP

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

Defines a mechanism and RDF vocabulary for describing the features supported by a SPARQL endpoint

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

Alternative formats used to serialize and exchange answers to SPARQL queries

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

SPARQL extension for executing queries distributed over different SPARQL endpoints

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

Extends SPARQL so that logically entailed RDF triples (hidden edges in RDF Graphs) are matched in addition to directly asserted RDF triples

What is SPARQL?

Components of SPARQL 1.1

- Query Language
- Update
- Protocol
- Service Description
- Query Results JSON Format
- Query Results CSV and TSV Format
- Query Results XML Format
- Federated Query
- Entailment Regimes
- Graph Store HTTP Protocol

Simple alternative to SPARQL 1.1 Update that describes HTTP operations for managing a collection of RDF graphs outside of a SPARQL 1.1 graph store

Agenda

1 What is SPARQL

2 SPARQL 1.1 Query Features by Example

3 Graph Patterns

4 Property Paths

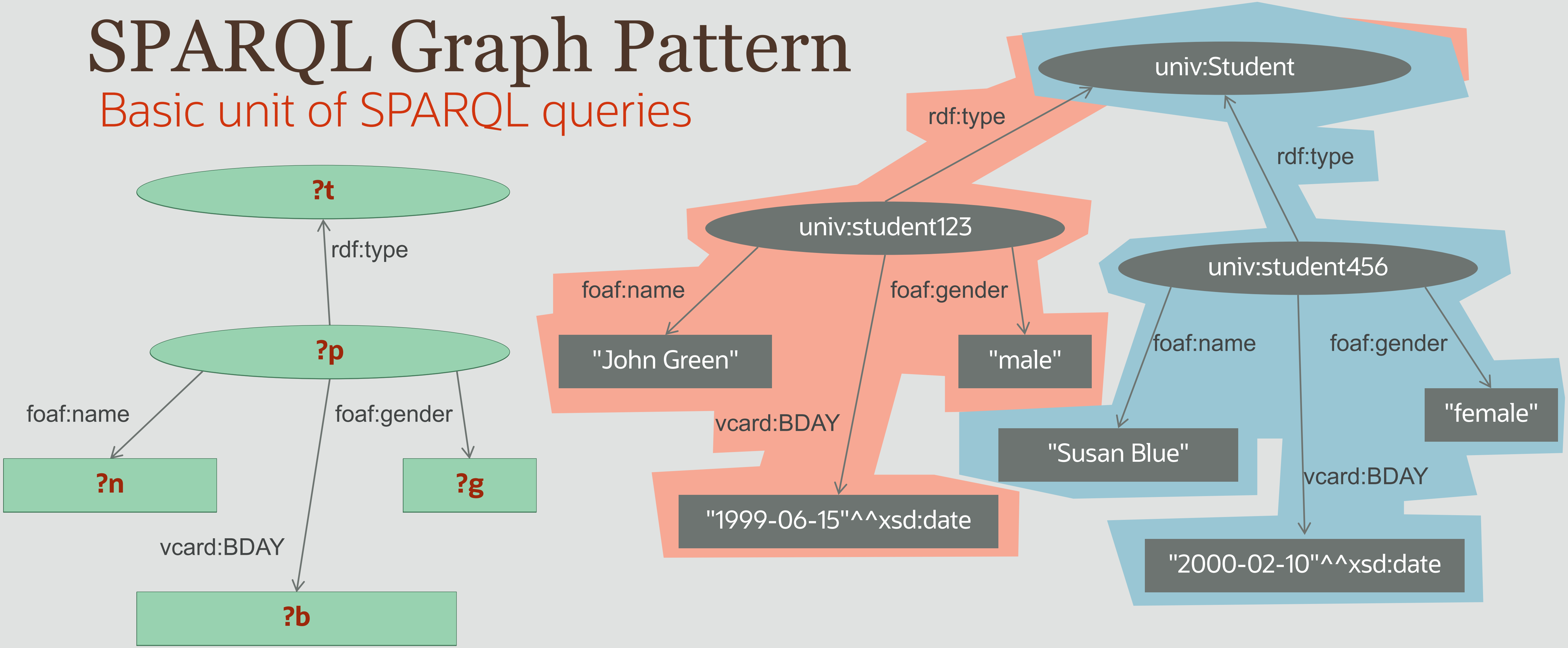
5 Named Graphs

6 Federated Queries

7 SPARQL Update

SPARQL Graph Pattern

Basic unit of SPARQL queries

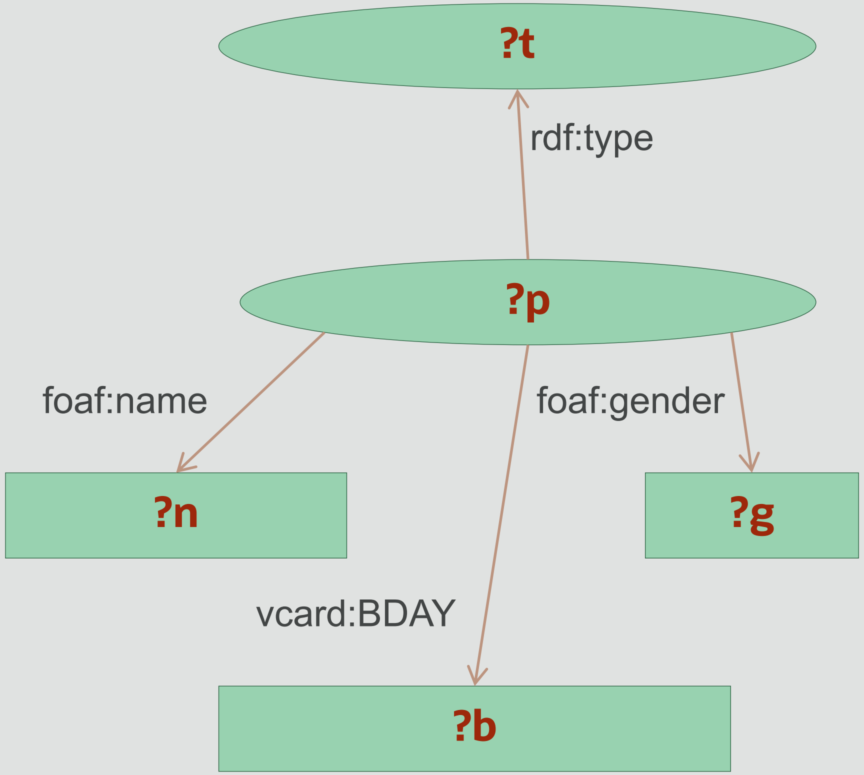


Result 1: {?t=univ:Student, ?p=univ:student123, ?n="John Green", ?g="male", ?b="1999-06-15"^^xsd:date}

Result 2: {?t=univ:Student, ?p=univ:student456, ?n="Susan Blue", ?g="female", ?b="2000-02-10"^^xsd:date}

SPARQL Graph Pattern

Basic unit of SPARQL queries



How do we express this with SPARQL?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?t ?n ?b ?g
WHERE
{ ?p rdf:type ?t .
  ?p foaf:name ?n .
  ?p vcard:BDAY ?b .
  ?p foaf:gender ?g }
```

Basic Graph Pattern (BGP)

SPARQL SELECT Modifiers

Find all DISTINCT genres of movies starring Keanu Reeves

```
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX movie: <http://data.linkedmdb.org/movie/>

SELECT DISTINCT ?gname
WHERE { ?movie movie:actor ?actor .
        ?actor movie:actor_name "Keanu Reeves" .
        ?movie movie:genre ?genre .
        ?genre movie:film_genre_name ?gname .
}
```

SPARQL FILTER: Restricting Solutions

Find movies starring Matt Damon that are more than 150 min long

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX movie: <http://data.linkedmdb.org/movie/>

SELECT ?title
WHERE {
  ?movie movie:actor ?actor .
  ?actor movie:actor_name ?aname .
  ?movie movie:runtime ?rt .
  ?movie dcterms:title ?title
  FILTER (?aname = "Matt Damon" && xsd:decimal(?rt) > 150)
}
```

SPARQL 1.1 Built-in Functions

Extensive library of functions to use

- **Basic:** arithmetic, comparisons, boolean connectors
- **RDF-related:** isLiteral(), isURI(), isBlank(), datatype(), lang(), BOUND(), …
- **String Functions:** SUBSTR(), STRSTARTS(), STRENDS(), REGEX(), …
- **Numerics:** abs(), floor(), ceil(), …
- **Dates and Times:** now(), year(), month(), day(), …
- **Miscellaneous:** IN(), NOT IN(), IF(), COALESCE(), …
- **Constructors:** xsd:int(), xsd:decimal(), xsd:dateTime(), …
- … plus user-defined

SPARQL UNION: Disjunction

Get names of writers and directors of movies starring Carl Weathers

```
SELECT ?name
WHERE {
  ?movie movie:actor ?actor .
  ?actor movie:actor_name "Carl Weathers" .
  { { ?movie movie:director ?director .
      ?director movie:director_name ?name }
    UNION
    { ?movie movie:writer ?writer .
      ?writer movie:writer_name ?name }
  }
}
```

SPARQL OPTIONAL: Best Effort Match

Find movies starring Sylvester Stallone and optionally their sequels

```
SELECT ?title ?title2
WHERE {
  ?movie dcterms:title ?title .
  ?movie movie:actor ?actor .
  ?actor movie:actor_name "Sylvester Stallone" .
  OPTIONAL {
    ?movie movie:sequel ?sequel .
    ?sequel dcterms:title ?title2
  }
}
```

Parallel vs. Nested OPTIONAL

RDF Data

```
:john foaf:name "John" ;
      foaf:email "john@example.com" ;
      foaf:homepage <http://www.example.com/john> .

:sue  foaf:name "Sue" ;
      foaf:email "sue@example.com" .

:fred  foaf:name "Fred" ;
      foaf:homepage <http://www.example.com/fred> .
```

Parallel OPTIONAL

```
{ ?s foaf:name ?n
   OPTIONAL { ?s foaf:email ?e }
   OPTIONAL { ?s foaf:homepage ?h }
}
```

Query Result

?s	?n	?e	?h
:john	"John"	"john@example.com"	<http://www.example.com/john>
:sue	"Sue"	"sue@example.com"	
:fred	"Fred"		<http://www.example.com/fred>

Parallel OPTIONAL:
Match all OPTIONALs from left to right.

Parallel vs. Nested OPTIONAL

RDF Data

```
:john foaf:name "John" ;
      foaf:email "john@example.com" ;
      foaf:homepage <http://www.example.com/john> .

:sue  foaf:name "Sue" ;
      foaf:email "sue@example.com" .

:fred  foaf:name "Fred" ;
      foaf:homepage <http://www.example.com/fred> .
```

Nested OPTIONAL

```
{ ?s foaf:name ?n
   OPTIONAL { ?s foaf:email ?e
     OPTIONAL { ?s foaf:homepage ?h }
   }
}
```

Query Result

?s	?n	?e	?h
:john	"John"	"john@example.com"	<http://www.example.com/john>
:sue	"Sue"	"sue@example.com"	
:fred	"Fred"		

Nested OPTIONAL:
Only match the child pattern if the parent matches.

SPARQL 1.1 Negation: MINUS

Movies starring Sylvester Stallone that do not have a sequel

```
SELECT ?title
WHERE {
  ?movie dcterms:title ?title .
  ?movie movie:actor ?actor .
  ?actor movie:actor_name "Sylvester Stallone" .
  MINUS {
    ?movie movie:sequel ?sequel .
  }
}
```

SPARQL 1.1 Negation: NOT EXISTS / EXISTS

Movies starring Robert De Niro that have a sequel

```
SELECT ?title
WHERE {
   ?movie dcterms:title ?title .
   ?movie movie:actor ?actor .
   ?actor movie:actor_name "Robert De Niro" .
   FILTER (EXISTS { ?movie movie:sequel ?sequel })
}
```

SPARQL Solution Modifiers: ORDER BY

Find all movies directed by Steven Spielberg ordered by ascending title and descending producer name

```
SELECT ?title ?pname
WHERE {
  ?movie dcterms:title ?title .
  ?movie movie:director ?director .
  ?director movie:director_name "Steven Spielberg" .
  ?movie movie:producer ?producer .
  ?producer movie:producer_name ?pname .
}
ORDER BY ASC(?title) DESC(?pname)
```

SPARQL Solution Modifiers: LIMIT / OFFSET

Find the 6th through 10th movies directed by Steven Spielberg

```
SELECT ?title ?rdate
WHERE {
  ?movie dcterms:title ?title .
  ?movie movie:director ?director .
  ?director movie:director_name "Steven Spielberg" .
  ?movie movie:initial_release_date ?rdate .
}
ORDER BY ASC(?rdate)
OFFSET 5
LIMIT 5
```

SPARQL 1.1 SELECT Expressions

Build a description string for a movie

```
SELECT (CONCAT(?title,
          " Released in ", ?rdate,
          " Directed by ", ?dname) AS ?mStr)
WHERE {
  ?movie dcterms:title ?title .
  ?movie movie:director ?director .
  ?director movie:director_name ?dname .
  ?movie movie:initial_release_date ?rdate .
}
LIMIT 10
```

SPARQL 1.1 Grouping and Aggregation

Find all director actor pairs for movies in the Star Wars series

```
SELECT ?dname ?aname
WHERE {
  ?movie dcterms:title ?title .
  ?movie movie:director ?director .
  ?director movie:director_name ?dname .
  ?movie movie:actor ?actor .
  ?actor movie:actor_name ?aname .
  ?movie movie:film_series ?series .
  ?series movie:film_series_name "Star Wars" .
}
GROUP BY ?dname ?aname
ORDER BY ?dname ?aname
```

SPARQL 1.1 Grouping and Aggregation

Find the 10 movie series with the most movies

```
SELECT ?sname (COUNT(?movie) AS ?mcnt)
WHERE {
  ?movie movie:film_series ?series .
  ?series movie:film_series_name ?sname .
}
GROUP BY ?sname
ORDER BY DESC(?mcnt)
LIMIT 10
```

Available Aggregates:

`COUNT(), SUM(), MIN(), MAX(), AVG(),`
`GROUP_CONCAT(), SAMPLE()`

SPARQL 1.1 Grouping and Aggregation

Find movie series having 3 or 4 movies

```
SELECT ?sname (COUNT(?movie) AS ?mcnt)
WHERE {
  ?movie movie:film_series ?series .
  ?series movie:film_series_name ?sname .
}
GROUP BY ?sname
HAVING (COUNT(?movie) IN (3,4))
ORDER BY DESC(?mcnt)
```

SPARQL 1.1 Subqueries

Find information about actors who have worked with more than 40 different directors

```
SELECT ?name
WHERE {
  { SELECT ?actor
    WHERE {
      ?movie movie:actor ?actor .
      ?movie movie:director ?director .
    }
    GROUP BY ?actor
    HAVING (COUNT(DISTINCT ?director) > 40)
  }
  ?actor movie:actor_name ?name .
}
```

SPARQL 1.1 Value Assignment: BIND

Find movies with a sequel named <title> II

```
SELECT ?title
WHERE {
   ?movie dcterms:title ?title .
   ?movie movie:sequel ?sequel .
   BIND (CONCAT(?title," II") AS ?part2)
   ?sequel dcterms:title ?part2
}
```

SPARQL 1.1 Inline Data: VALUES

Find Action Movies with Uma Thurman and Comedy Movies with John Candy

```
SELECT ?aname ?title
WHERE { ?movie dcterms:title ?title .
        ?movie movie:actor ?actor .
        ?actor movie:actor_name ?aname .
        ?movie movie:genre ?genre .
        ?genre movie:film_genre_name ?gname .
  VALUES (?aname ?gname) { ("Uma Thurman" "Action")
                           ("John Candy" "Comedy") }
}
```

SPARQL ASK Queries

Has Danny DeVito acted in an Action movie?

```
ASK
WHERE { ?movie movie:actor ?actor .
        ?actor movie:actor_name "Danny DeVito" .
        ?movie movie:genre ?genre .
        ?genre movie:film_genre_name "Action" .
}
```

SPARQL Construct Queries

Build a co-star graph

```
CONSTRUCT { ?actor1 movie:co_star ?actor2 }
WHERE { ?movie movie:actor ?actor1 .
        ?movie movie:actor ?actor2 .
        FILTER (!sameTerm(?actor1, ?actor2))
}
LIMIT 50
```

SPARQL Describe Queries

Describe a single resource

```
DESCRIBE <http://data.linkedmdb.org/film/37164>
```

SPARQL Describe Queries

Describe variables in a bigger query

```
DESCRIBE ?director
WHERE { ?movie dcterms:title "Toy Story" .
        ?movie movie:director ?director
}
```

Agenda

1 What is SPARQL

2 SPARQL 1.1 Query Features by Example

3 Graph Patterns

4 Property Paths

5 Named Graphs

6 Federated Queries

7 SPARQL Update

SPARQL 1.1 Property Paths

Enhanced path searching in SPARQL

- Uses regular expression style syntax to express path patterns over RDF properties
- Allows syntactic shortcuts for fixed length paths
- Allows searching arbitrary length paths
- Computes reachability rather than enumerating paths

Property Path Constructs

<table>
<tr><th>Syntax Form</th><th>Matches</th></tr>
<tr><td>iri</td><td>An IRI (path of length 1)</td></tr>
<tr><td>^elt</td><td>Reverse path (object to subject)</td></tr>
<tr><td>elt1 / elt2</td><td>Sequence path of elt1 followed by elt2</td></tr>
<tr><td>elt1 | elt2</td><td>Alternative path of elt1 or elt2</td></tr>
<tr><td>elt*</td><td>Path composed of zero or more repetitions of elt</td></tr>
<tr><td>elt+</td><td>Path composed of one or more repetitions of elt</td></tr>
<tr><td>elt?</td><td>Path composed of zero or one repetition of elt</td></tr>
<tr><td>!iri or $!(iri_1|iri_2|\ldots|iri_n)$</td><td>A path of length 1 that is not one of iri_i</td></tr>
<tr><td>!^iri or $!(\hat{}iri_1|\hat{}iri_2|\ldots|\hat{}iri_n)$</td><td>A path of length 1 that is not one of iri_i as reverse paths</td></tr>
<tr><td>$!(iri_1|\ldots|iri_j|\hat{}iri_{j+1}|\ldots|\hat{}iri_n)$</td><td>A path of length 1 that is not one of iri_i in the indicated direction</td></tr>
<tr><td>(elt)</td><td>Grouping used to control precedence</td></tr>
</table>

iri **is an IRI**

elt **is a path element, which may itself be composed of other path constructs**

SPARQL 1.1 Property Path

Find all sequels for The Terminator

```
SELECT ?stitle
WHERE { ?movie dcterms:title "The Terminator" .
        ?movie movie:sequel+ ?sequel .
        ?sequel dcterms:title ?stitle
}
```

SPARQL 1.1 Property Path

Get names of writers and directors of movies starring Carl Weathers

```
SELECT ?name
WHERE {
  ?movie movie:actor ?actor .
  ?actor movie:actor_name "Carl Weathers" .
  { { ?movie movie:director ?director .
      ?director movie:director_name ?name }
    UNION
    { ?movie movie:writer ?writer .
      ?writer movie:writer_name ?name }
  }
}
```

SPARQL 1.1 Property Path

Get names of writers and directors of movies starring Carl Weathers

Simplified with property path syntactic sugar

```
SELECT ?name
WHERE {
  ?movie movie:actor/movie:actor_name "Carl Weathers" .
  ?movie (movie:director/movie:director_name)|
         (movie:writer/movie:writer_name) ?name .}
```

Agenda

1 What is SPARQL

2 SPARQL 1.1 Query Features by Example

3 Graph Patterns

4 Property Paths

5 Named Graphs

6 Federated Queries

7 SPARQL Update

SPARQL Named Graphs

The concept of an *RDF Dataset*

- An *RDF Dataset* is a collection of RDF graphs
  - Contains one *default graph*, which does not have a name
  - Contains zero or more *named graphs*, where each graph is identified by an IRI
- A SPARQL query is executed against an RDF Dataset
- **FROM** and **FROM NAMED** keywords are used to construct the RDF Dataset for a query
- The **GRAPH** keyword is used to control the *active graph* for different parts of a query

Constructing the RDF Dataset

Contents of RDF Triplestore

Graph Name	Triples
--	{t1,t2,t3}
<urn:g1>	{t4,t5}
<urn:g2>	{t6,t7}
<urn:g3>	{t8,t9}
<urn:g4>	{t10,t11}

SPARQL query with RDF Dataset specification

```
SELECT *
FROM <urn:g1>
FROM <urn:g3>
FROM NAMED <urn:g2>
FROM NAMED <urn:g3>
FROM NAMED <urn:g4>
WHERE { … }
```

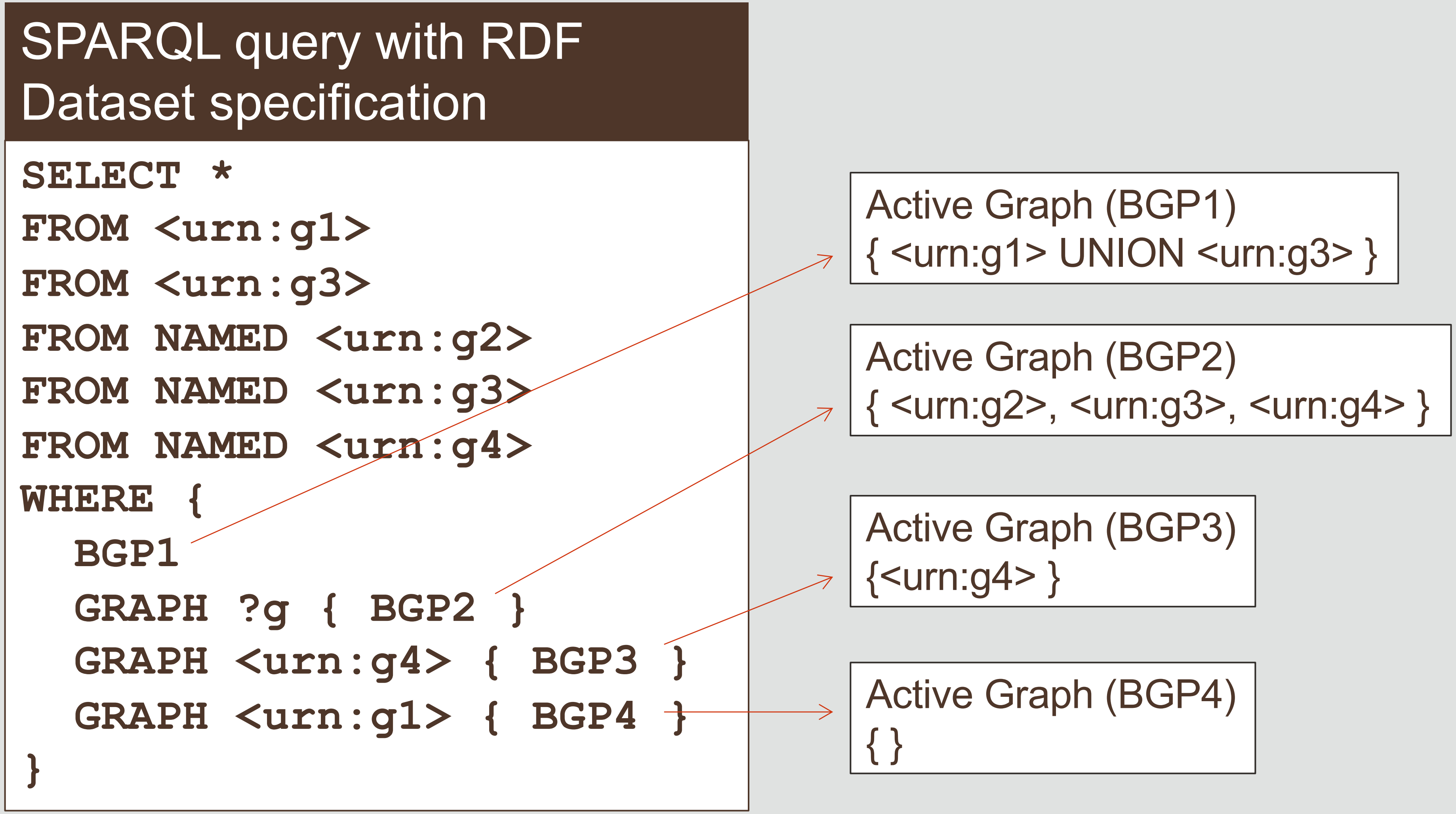
Default Graph

{ t4, t5, t8, t9 }

Named Graphs

{ (<urn:g2>, { t6, t7 }),
(<urn:g3>, { t8, t9 }),
(<urn:g4>, { t10, t11 }) }

Using the GRAPH Keyword



SPARQL query with RDF Dataset specification

```
SELECT *
FROM <urn:g1>
FROM <urn:g3>
FROM NAMED <urn:g2>
FROM NAMED <urn:g3>
FROM NAMED <urn:g4>
WHERE {
  BGP1
  GRAPH ?g { BGP2 }
  GRAPH <urn:g4> { BGP3 }
  GRAPH <urn:g1> { BGP4 }
}
```

Active Graph (BGP1)
{ <urn:g1> UNION <urn:g3> }

Active Graph (BGP2)
{ <urn:g2>, <urn:g3>, <urn:g4> }

Active Graph (BGP3)
{<urn:g4> }

Active Graph (BGP4)
{ }

Within a **GRAPH** clause:

- BGP is executed against each active graph separately (e.g. BGP2 against g2, g3, g4).

- Subgraph match must occur within a single graph.

SPARQL Named Graph Query

Find the number of bills sponsored by each politician in the 110th and 111th congress

```
SELECT ?n ?g (count(?b) as ?bcnt)
FROM usgov:people
FROM NAMED usgov:bills_110
FROM NAMED usgov:bills_111
WHERE
{ ?s foaf:name ?n
  GRAPH ?g { ?b bill:sponsor ?s }
}
GROUP BY ?n ?g
ORDER BY ?n ?g
```

SPARQL Named Graph Query

Edge Property: Find critics and their ratings for The Matrix

```
SELECT ?cname ?r
WHERE { ?movie dcterms:title "The Matrix" .
        GRAPH ?review { ?critic movie:reviewed ?movie .}
        ?review movie:rating ?r
        ?critic movie:critic_name ?cname
}
```

Agenda

1 What is SPARQL

2 SPARQL 1.1 Query Features by Example

3 Graph Patterns

4 Property Paths

5 Named Graphs

6 Federated Queries

7 SPARQL Update

SPARQL 1.1 Federated Query

- Used to execute a single query over multiple, possibly distributed RDF datasources
- Portions of a query can be directed to particular SPARQL endpoints
- Results are returned to the federated query processor and combined with the rest of the query

SPARQL 1.1 Federated Query

Find birth year, child and spouse information from DBPedia for Tom Hanks

```
SELECT ?a ?dbpUri ?byear ?child ?spouse
WHERE {
  ?a movie:actor_name "Tom Hanks";
     owl:sameAs ?dbpUri .
  FILTER (STRSTARTS(STR(?dbpUri),"http://dbpedia.org"))
  SERVICE <http://dbpedia.org/sparql> {
    ?dbpUri dbo:birthYear ?byear ;
            dbo:child     ?child ;
            dbo:spouse    ?spouse .
   }
}
```

Agenda

1 What is SPARQL

2 SPARQL 1.1 Query Features by Example

3 Graph Patterns

4 Property Paths

5 Named Graphs

6 Federated Queries

7 SPARQL Update

SPARQL 1.1 Update

Capabilities of SPARQL Update

- Insert triples into an RDF Graph
- Delete triples from an RDF Graph
- Load an RDF Graph
- Clear an RDF Graph
- Create a new RDF Graph
- Drop an RDF Graph
- Copy, move or add the content of one RDF Graph to another
- Perform a group of update operations as a single action

SPARQL 1.1 Update

Example – INSERT DATA

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA {
     <http://example/book1> dc:title "A new book" ;
                                    dc:creator "A.N.Other" .  }
```

Constant quad pattern

Data before:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book1> ns:price 42 .
```

Data after:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book1> ns:price 42 .
<http://example/book1> dc:title "A new book" .
<http://example/book1> dc:creator "A.N.Other" .
```

SPARQL 1.1 Update

Example – DELETE DATA

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
DELETE DATA {
     <http://example/book2> dc:title "David Copperfield" ;
                                         dc:creator "Edmund Wells" . }
```

Data before:

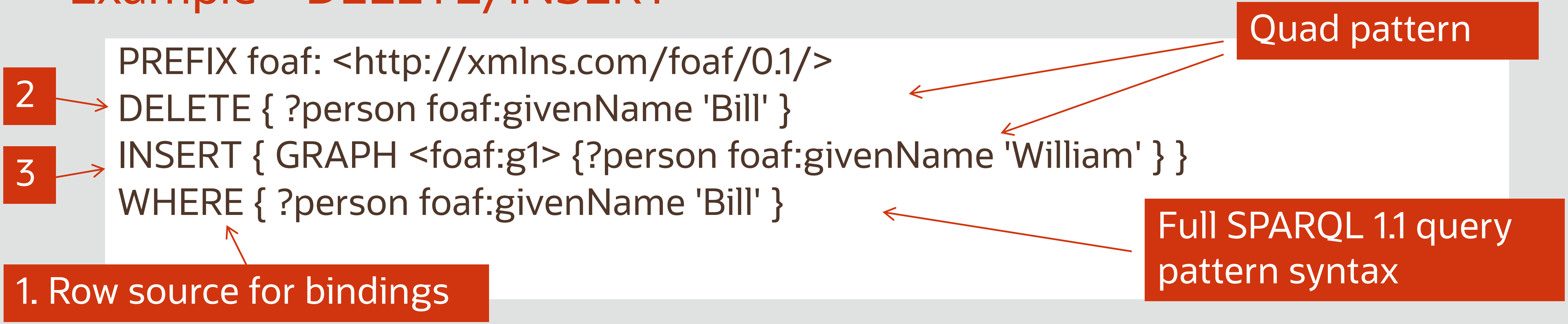
```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book2> ns:price 42 .
<http://example/book2> dc:title "David Copperfield" .
<http://example/book2> dc:creator "Edmund Wells" .
```

Data after:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .
<http://example/book2> ns:price 42 .
```

SPARQL 1.1 Update

Example – DELETE/INSERT



```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DELETE { ?person foaf:givenName 'Bill' }
INSERT { GRAPH <foaf:g1> {?person foaf:givenName 'William' } }
WHERE { ?person foaf:givenName 'Bill' }
```

Quad pattern
2
3
Full SPARQL 1.1 query pattern syntax
1. Row source for bindings

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<http://example/president27> foaf:givenName "Bill" .
<http://example/president27> foaf:familyName "Taft" .
<http://example/president42> foaf:givenName "Bill" .
<http://example/president42> foaf:familyName
"Clinton" .
```

Data after:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
foaf:g1 {
   <http://example/president27> foaf:givenName "William" .
   <http://example/president42> foaf:givenName "William" .
}
<http://example/president27> foaf:familyName "Taft" .
<http://example/president42> foaf:familyName "Clinton" .
```

SPARQL 1.1 Update

Example – LOAD

```
LOAD <http://example.com/addresses>
INTO GRAPH <http://example.com/addresses>
```

GRAPH <URI>

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
```

Data after:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
   <http://example/fred> foaf:mbox <mailto:fred@example> .
}
```

SPARQL 1.1 Update

Example – CLEAR

```
CLEAR GRAPH <http://example.com/addresses>
```

GRAPH <URI>
or
DEFAULT
or
NAMED
or
ALL

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
   <http://example/fred> foaf:mbox <mailto:fred@example> .
}
```

Data after:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
```

SPARQL 1.1 Update

Example – CREATE

```
CREATE GRAPH <http://example.com/addresses>
```

Data before:

Data after:

```
# Graph: http://example.com/addresses
```

SPARQL 1.1 Update

Example – DROP

```
DROP GRAPH <http://example.com/addresses>
```

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
   <http://example/fred> foaf:mbox <mailto:fred@example> .
}
```

Data after:

SPARQL 1.1 Update

Example – COPY

```
COPY GRAPH <http://example.com/addresses>
TO GRAPH <http://example.com/addresses2>
```

Data before:	Data after:

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
# Graph: http://example.com/addresses2
ex:addresses2 {
   <http://example/fred> foaf:mbox <mailto:fred@example> .
}
```

Data after:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
# Graph: http://example.com/addresses2
ex:addresses2 {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
```

SPARQL 1.1 Update

Example – MOVE

```
MOVE GRAPH <http://example.com/addresses>
TO GRAPH <http://example.com/addresses2>
```

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
# Graph: http://example.com/addresses2
ex:addresses2 {
   <http://example/fred> foaf:mbox <mailto:fred@example> .
}
```

Data after:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses2
ex:addresses2 {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
```

SPARQL 1.1 Update

Example – ADD

```
ADD GRAPH <http://example.com/addresses>
TO GRAPH <http://example.com/addresses2>
```

Data before:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
# Graph: http://example.com/addresses2
ex:addresses2 {
   <http://example/fred> foaf:mbox <mailto:fred@example> .
}
```

Data after:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix ex: <http://example.com/> .
# Graph: http://example.com/addresses
ex:addresses {
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
# Graph: http://example.com/addresses2
ex:addresses2 {
   <http://example/fred> foaf:mbox <mailto:fred@example> .
   <http://example/bill> foaf:mbox <mailto:bill@example> .
}
```

SPARQL 1.1 Update

Transaction Support

```
INSERT { ?s :fullName ?name }
WHERE {
  SELECT ?s (CONCAT(?fname, " ", ?lname) AS ?name)
  WHERE { ?s :fname ?fname;
                      :lname ?lname }
};
DELETE { ?s :mbox ?mail }
INSERT { ?s :email ?mail }
WHERE { ?s :mbox ?mail };
DELETE DATA { :emp1 :phone "603-123-4567" . }
```

A sequence of updates should run as a **single transaction**

Agenda

1 Graph Query Languages

2 Essentials for SPARQL Query & Update

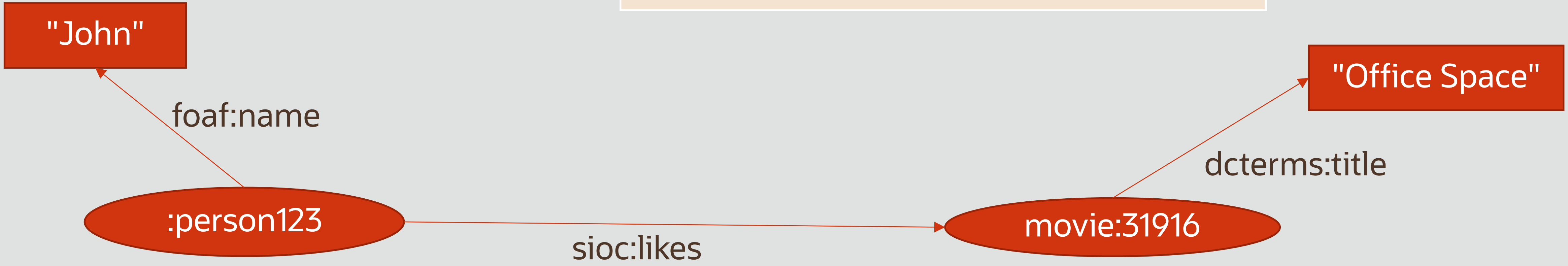
3 Named Graphs for Edge Properties

4 Comparison with PG Query Languages

5 Graph Analytics with RDF Data

Adding Movie Reviews

Edge
John likes Office Space



Adding Movie Reviews

John likes Office Space

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX         : <http://example.com/data/>

INSERT DATA {
  # John likes Office Space
  :person123 foaf:name "John" ;
             sioc:likes movie:3196 .
}
```

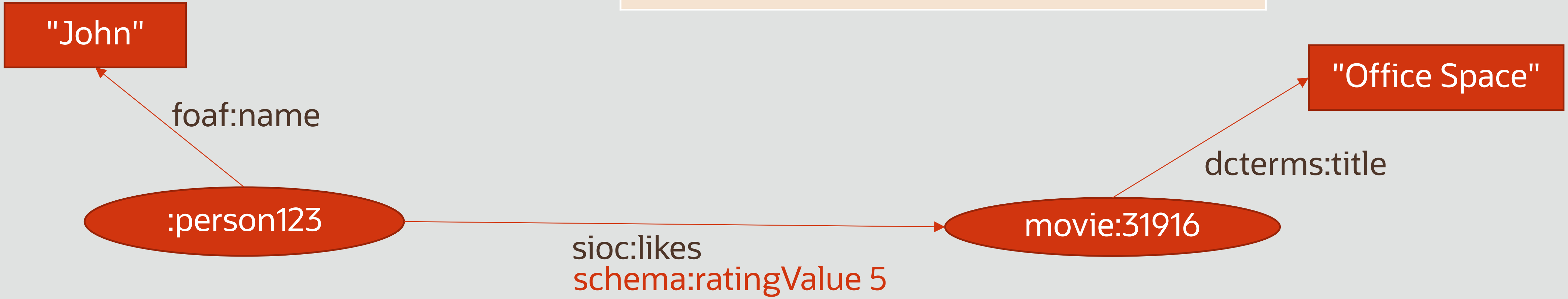
Adding Movie Reviews

Who likes Office Space?

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX         : <http://example.com/data/>

SELECT ?name
WHERE {
   ?person foaf:name ?name .
   ?person sioc:likes ?movie .
   ?movie dcterms:title "Office Space" .
}
```

Adding Movie Reviews

Edge Property
John likes Office Space with a rating of 5



Adding Movie Reviews

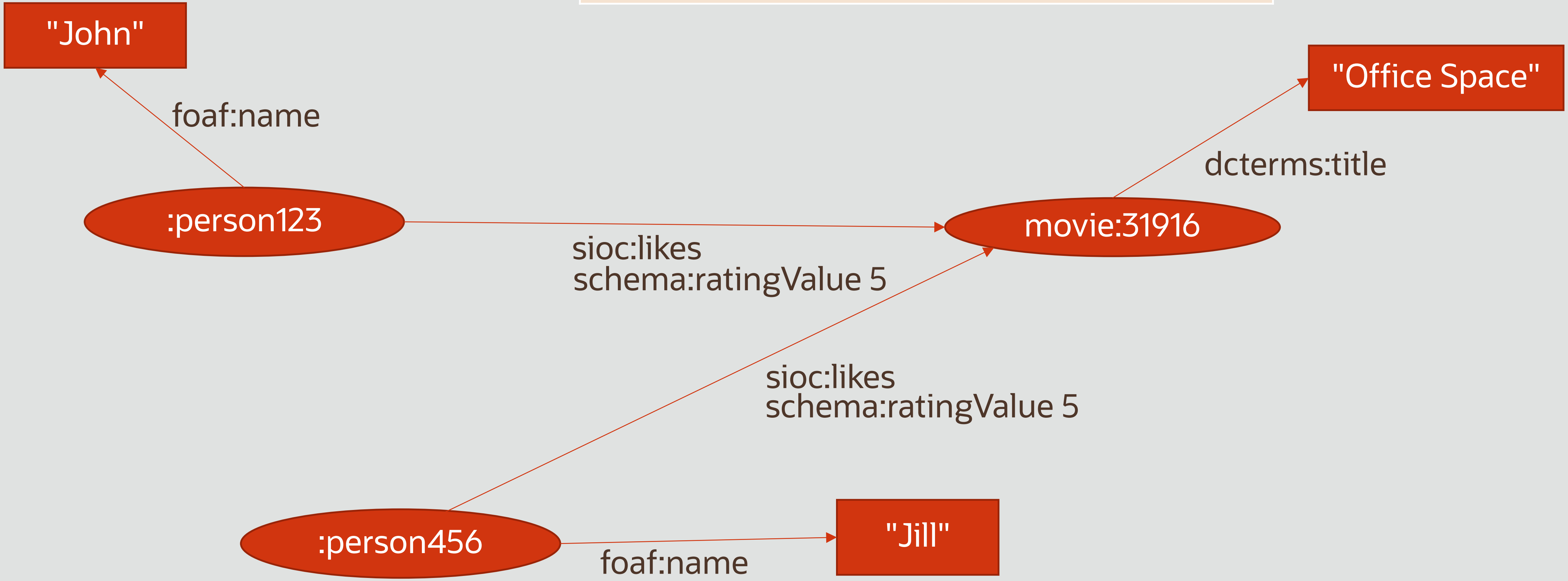
John likes Office Space with a rating of 5

```
PREFIX     movie: <http://data.linkedmdb.org/movie/>
PREFIX       foaf: <http://xmlns.com/foaf/0.1/>
PREFIX   dcterms: <http://purl.org/dc/terms/>
PREFIX        sioc: <http://rdfs.org/sioc/ns#>
PREFIX    schema: <http://data.linkedmdb.org/movie/>
PREFIX               : <http://example.com/data/>

# remove triple
DELETE DATA { :person123 sioc:likes movie:31916 . }
INSERT DATA {
   # replace triple with quad assigning :edge1 as id
   GRAPH :edge1 { :person123 sioc:likes movie:3196 . }
   # add edge property for rating
   :edge1 schema:ratingValue 5 .
}
```

Adding Movie Reviews

Edge Property

Jill also likes Office Space with a rating of 5



Adding Movie Reviews

Jill also likes Office Space with a rating of 5

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX         : <http://example.com/data/>

INSERT DATA {
   # add Jill
   :person456 foaf:name "Jill" .
   # edge id of :edge2 for Jill likes Office Space
   GRAPH :edge2 { :person456 sioc:likes movie:3196 . }
   # add edge property for rating
   :edge2 schema:ratingValue 5 .
}
```

Adding Movie Reviews

Find ratings for Office Space

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX         : <http://example.com/data/>

SELECT ?name ?rating
WHERE {
   ?movie dcterms:title "Office Space" .
   GRAPH ?edge { ?person sioc:likes ?movie }
   ?person foaf:name ?name .
   ?edge schema:ratingValue ?rating .
}
```

Adding Movie Reviews

Who likes Office Space?

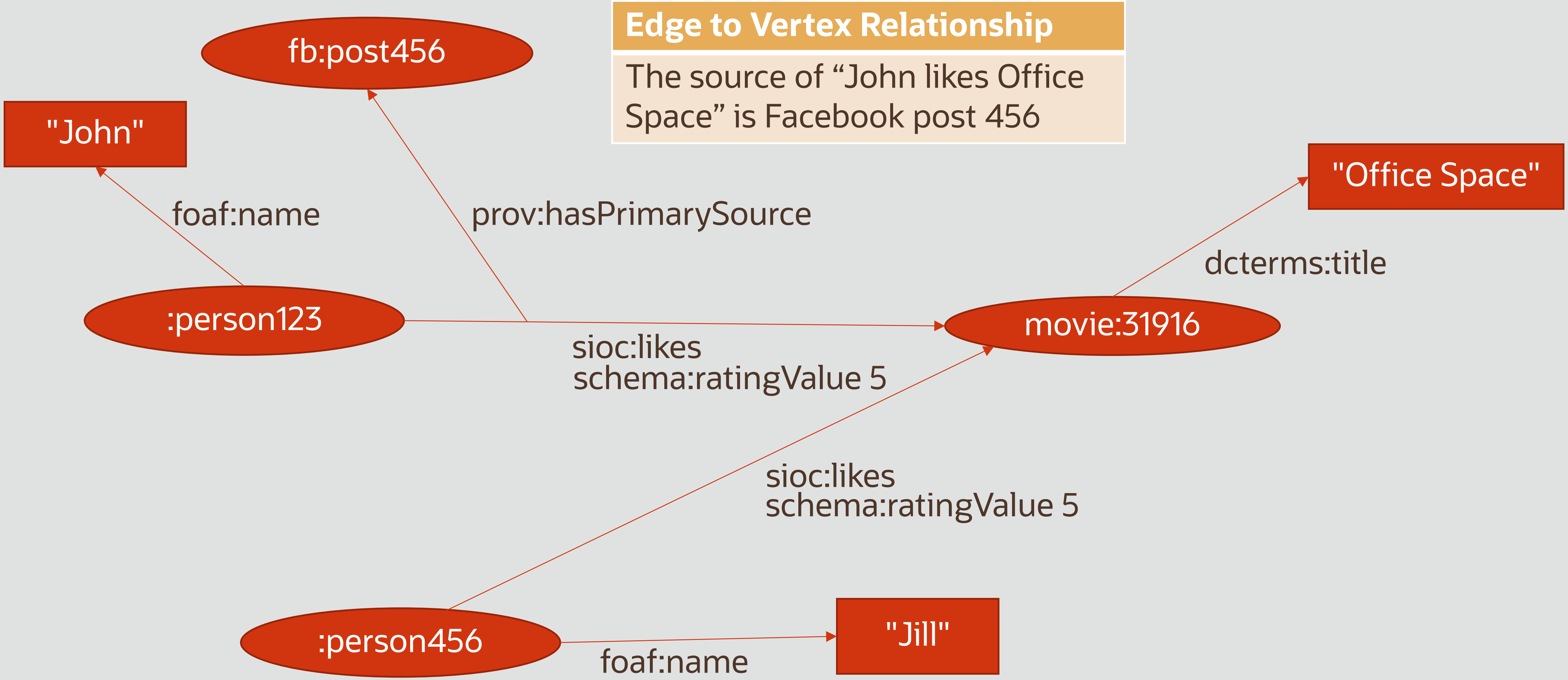
```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX      foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX      sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX             : <http://example.com/data/>

SELECT ?name
WHERE {
   ?person foaf:name ?name .
   ?person sioc:likes ?movie .
   ?movie dcterms:title "Office Space" .
}
```

Old queries still work!

Adding Movie Reviews

Edge to Vertex Relationship

The source of “John likes Office Space” is Facebook post 456



Adding Movie Reviews

The source of “John likes Office Space” is Facebook post 456

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX       fb: <http://www.facebook.com/>
PREFIX     prov: <http://www.w3.org/ns/prov#>
PREFIX         : <http://example.com/data/>

INSERT DATA {
   # add source information for :edge1
   :edge1 prov:hadPrimarySource fb:post456 .
}
```

Adding Movie Reviews

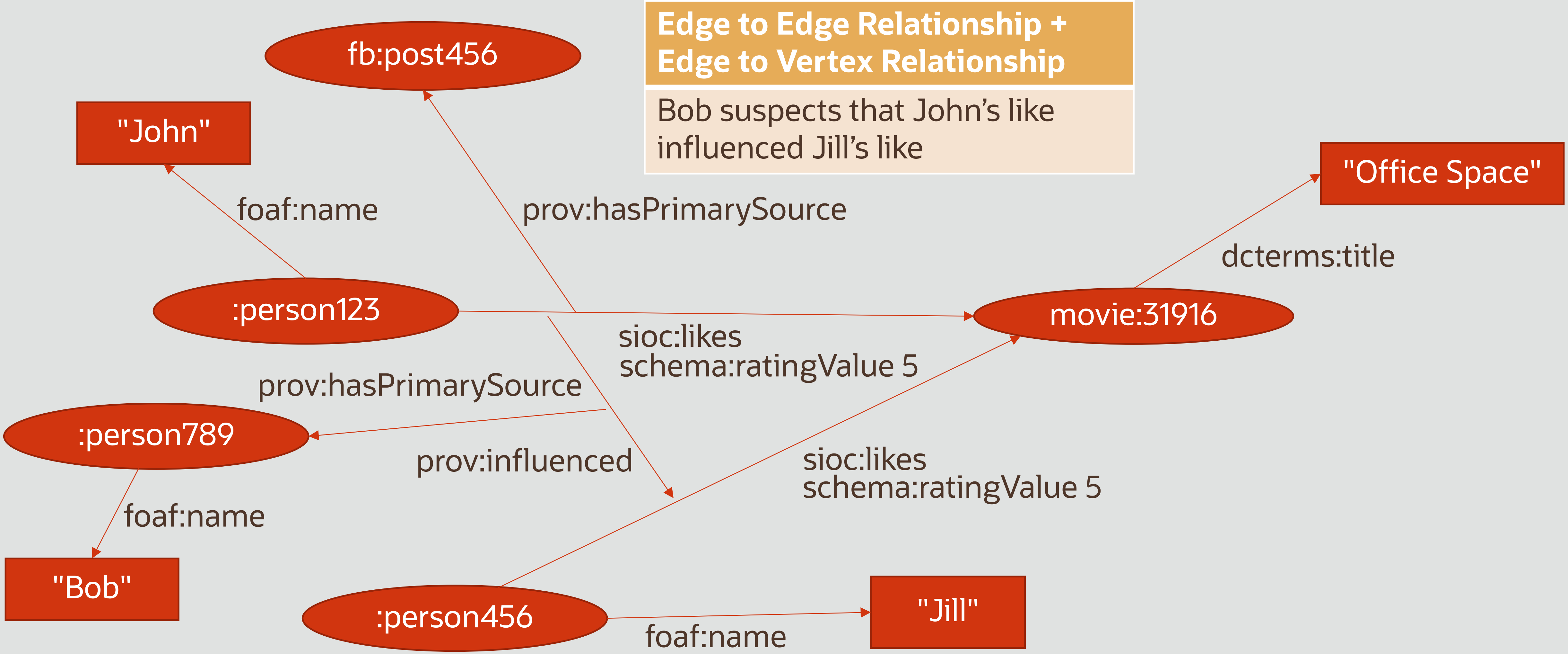
What is the source of “John likes Office Space”?

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX       fb: <http://www.facebook.com/>
PREFIX     prov: <http://www.w3.org/ns/prov#>
PREFIX         : <http://example.com/data/>

SELECT ?source
WHERE {
   ?person foaf:name "John" .
   GRAPH ?edge { ?person sioc:likes ?movie }
   ?movie dcterms:title "Office Space" .
   ?edge prov:hadPrimarySource ?source .
}
```

Adding Movie Reviews

Edge to Edge Relationship + Edge to Vertex Relationship

Bob suspects that John's like influenced Jill's like



Adding Movie Reviews

Bob suspects that John’s like influenced Jill’s like

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX       foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX       sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX          fb: <http://www.facebook.com/>
PREFIX       prov: <http://www.w3.org/ns/prov#>
PREFIX             : <http://example.com/data/>

INSERT DATA {
   # add Bob
   :person789 foaf:name "Bob" .
   # edge id of :edge3 for influenced
   GRAPH :edge3 { :edge1 prov:influenced :edge2 . }
   # Bob is the source of the influenced edge
   :edge3 prov:hasPrimarySource :person789 .
}
```

Adding Movie Reviews

Who suspects that John’s like influenced Jill’s like

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX     prov: <http://www.w3.org/ns/prov#>
PREFIX         : <http://example.com/data/>

SELECT ?person
WHERE {
   ?john foaf:name "John" .
   GRAPH ?edge1 { ?john sioc:likes ?movie }
   ?movie dcterms:title "Office Space" .
   ?jill foaf:name "Jill" .
   GRAPH ?edge2 { ?jill sioc:likes ?movie }
   GRAPH ?edge3 { ?edge1 prov:influenced ?edge2 }
   ?edge3 prov:hasPrimarySource/foaf:name ?person .
}
```

Adding Movie Reviews

Who likes Office Space?

```
PREFIX    movie: <http://data.linkedmdb.org/movie/>
PREFIX     foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dcterms: <http://purl.org/dc/terms/>
PREFIX     sioc: <http://rdfs.org/sioc/ns#>
PREFIX   schema: <http://data.linkedmdb.org/movie/>
PREFIX         : <http://example.com/data/>

SELECT ?name
WHERE {
   ?person foaf:name ?name .
   ?person sioc:likes ?movie .
   ?movie dcterms:title "Office Space" .
}
```

Old queries still work!

Agenda

1 Graph Query Languages

2 Essentials for SPARQL Query & Update

3 Named Graphs for Edge Properties

4 Comparison with PG Query Languages

5 Graph Analytics with RDF Data

Graph Query Languages

RDF Graph

SPARQL 1.1

SPARQL 1.2
SPARQL*

Property Graph

PGQL

G-CORE

Cypher

SQL/PGQ
GQL

GSQL

Gremlin

Property Graph Query Languages

- PG query language design aligns more with graph as a data structure rather than RDF triple/quad
  - More features for path searching and graph algorithms
    - Shortest path, k-shortest path, inDegree(), outDegree(), …
  - Use “ASCII-art” for edge pattern expression
    - (a:person)-[e:knows]->(b:person)
  - Vertices and Edges are objects with properties
- SPARQL has more features for data integration use cases
  - Standard Protocol
  - OPTIONAL patterns
  - Federated Query
  - Entailment Regimes

PGQL Graph Query Language

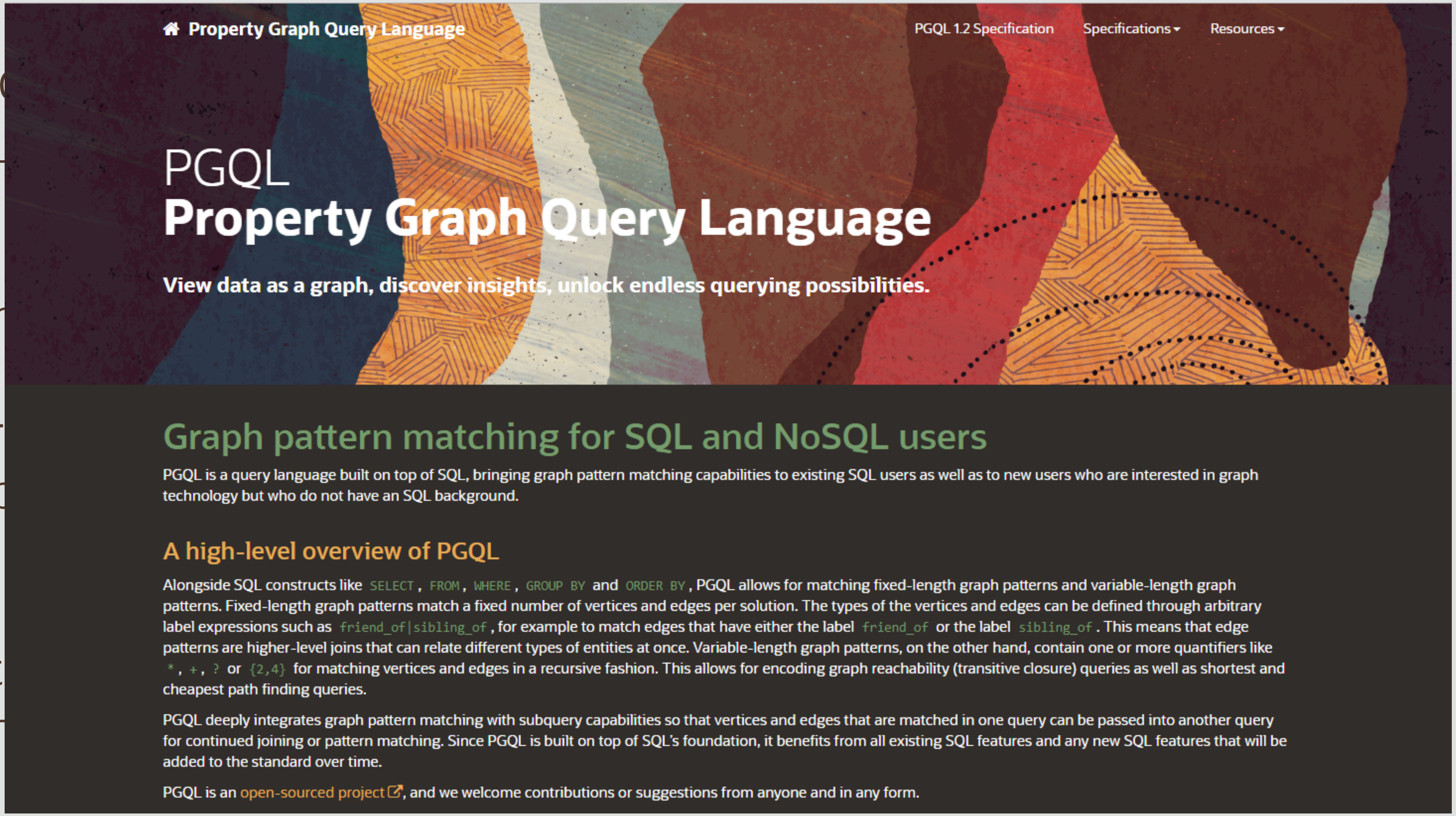
pgql-lang.org

Graph pattern mat
(person) –[:works_

Basic patterns and
Can we reach from

Shortest path quer
Find the shortest p

Familiarity for SQL
Similar language c
SELECT … WH
“Result set” (table)



PGQL
Property Graph Query Language

View data as a graph, discover insights, unlock endless querying possibilities.

Graph pattern matching for SQL and NoSQL users

PGQL is a query language built on top of SQL, bringing graph pattern matching capabilities to existing SQL users as well as to new users who are interested in graph technology but who do not have an SQL background.

A high-level overview of PGQL

Alongside SQL constructs like `SELECT`, `FROM`, `WHERE`, `GROUP BY` and `ORDER BY`, PGQL allows for matching fixed-length graph patterns and variable-length graph patterns. Fixed-length graph patterns match a fixed number of vertices and edges per solution. The types of the vertices and edges can be defined through arbitrary label expressions such as `friend_of|sibling_of`, for example to match edges that have either the label `friend_of` or the label `sibling_of`. This means that edge patterns are higher-level joins that can relate different types of entities at once. Variable-length graph patterns, on the other hand, contain one or more quantifiers like `*`, `+`, `?` or `{2,4}` for matching vertices and edges in a recursive fashion. This allows for encoding graph reachability (transitive closure) queries as well as shortest and cheapest path finding queries.

PGQL deeply integrates graph pattern matching with subquery capabilities so that vertices and edges that are matched in one query can be passed into another query for continued joining or pattern matching. Since PGQL is built on top of SQL's foundation, it benefits from all existing SQL features and any new SQL features that will be added to the standard over time.

PGQL is an open-sourced project, and we welcome contributions or suggestions from anyone and in any form.

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX   : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT p2.name AS friend, u.name AS university
MATCH (p1:Person) -[:knows]-> (p2:Person) ,
           (p1) -[:studentOf]-> (u:University) ,
           (p2) -[:studentOf]-> (u)
WHERE p1.name = 'Lee'
```

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX   : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University; :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT p2.name AS friend, u.name AS university
MATCH (p1:Person) -[:knows]-> (p2:Person) ,
          (p1) -[:studentOf]-> (u:University) ,
          (p2) -[:studentOf]-> (u)
WHERE p1.name = 'Lee'
```

Identifiers for resources, classes, types are Strings (labels) not URIs

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX  : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
   ?p1 a :Person;  :studentName ?p1Name .
   ?p2 a :Person;  :studentName ?p2Name .
   ?u  a :University;  :universityName ?univName .
   ?p1 :studentOf ?u .
   ?p1 :knows ?p2 .
   ?p2 :studentOf ?u .
   FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT p2.name AS friend, u.name AS university
MATCH (p1:Person) -[:knows]-> (p2:Person) ,
      (p1) -[:studentOf]-> (u:University) ,
      (p2) -[:studentOf]-> (u)
WHERE p1.name = 'Lee'
```

Variables are not prefixed with a ‘?’. Syntax rules used to identify variables.

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX  : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT p2.name AS friend, u.name AS university
MATCH (p1:Person) -[:knows]-> (p2:Person) ,
      (p1) -[:studentOf]-> (u:University) ,
      (p2) -[:studentOf]-> (u)
WHERE p1.name = 'Lee'
```

Edge traversals are specified with ASCII art instead of triple patterns

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX   : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT p2.name AS friend, u.name AS university
MATCH (p1:Person) -[:knows]-> (p2:Person) ,
          (p1) -[:studentOf]-> (u:University) ,
          (p2) -[:studentOf]-> (u)
WHERE p1.name = 'Lee'
```

Vertex type information is specified with a label constraint instead of rdf:type triples.

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX  : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT p2.name AS friend, u.name AS university
MATCH (p1:Person) -[:knows]-> (p2:Person) ,
            (p1) -[:studentOf]-> (u:University) ,
            (p2) -[:studentOf]-> (u)
WHERE p1.name = 'Lee'
```

Edge type information is specified with a label constraint instead of predicate URI

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX   : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT  p2.name AS friend, u.name AS university
MATCH  (p1:Person) -[:knows]-> (p2:Person) ,
             (p1) -[:studentOf]-> (u:University) ,
             (p2) -[:studentOf]-> (u)
WHERE  p1.name = 'Lee'
```

Vertex properties are specified as attributes with dot notation instead of with triple patterns and variables.

PGQL for SPARQL Users

“Find people that Lee knows and that are a student at the same university as Lee”

SPARQL 1.1

```
PREFIX  : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

PGQL 1.2

```
SELECT  p2.name AS friend, u.name AS university
MATCH  (p1:Person) -[:knows]-> (p2:Person) ,
          (p1) -[:studentOf]-> (u:University) ,
          (p2) -[:studentOf]-> (u)
WHERE  p1.name = 'Lee'
```

Projection and filter expressions are similar

PGQL for SPARQL Users

Specifying everything as triples can be verbose, but …

SPARQL 1.1

```
PREFIX   : <http://univ/vocab#>
SELECT (?p2Name AS ?friend) (?univName AS ?university)
WHERE {
    ?p1 a :Person;  :studentName ?p1Name .
    ?p2 a :Person;  :studentName ?p2Name .
    ?u   a :University;  :universityName ?univName .
    ?p1 :studentOf ?u .
    ?p1 :knows ?p2 .
    ?p2 :studentOf ?u .
    FILTER (?p1Name = "Lee")
}
```

Allows discovery of schema:
What edge types and property types are available?

```
SELECT DISTINCT ?p
WHERE { ?s ?p ?o  . }
```

What vertex types are available?

```
SELECT DISTINCT ?t
WHERE { ?s rdf:type ?t  . }
```

Works well for irregular data:
Project all properties for each Student

```
SELECT ?s ?p ?o
WHERE {
   ?s a :Student ;
   ?s ?p ?o .
}
```

Path Searching in PGQL and SPARQL

Reachability: Is Lee connected to Tom through a sequence of knows relations?

SPARQL 1.1

```
PREFIX  : <http://univ/vocab#>
SELECT ("yes" AS ?isConnected)
WHERE {
    ?p1 :studentName "Lee" .
    ?p2 :studentName "Tom" .
    ?p1 :knows+ ?p2 .
}
```

PGQL 1.2

```
SELECT 'yes' AS isConnected
MATCH (p1:Person) -/:knows+/-> (p2:Person)
WHERE p1.name = 'Lee' AND p2.name = 'Tom'
```

Both query languages use regex-style syntax for one or more and zero or more. PGQL uses /p/ instead of [p] to specify reachability

Path Searching in PGQL and SPARQL

Shortest Path: Find the shortest path connecting Lee to Tom through a sequence of knows relations

SPARQL 1.1

Not Possible

PGQL 1.2

```
SELECT COUNT(e) AS pathLen,
            ARRAY_AGG(b.name) AS friends
MATCH SHORTEST ( (p1:Person) ((a) –[e:knows]-> (b))* (p2:Person) )
WHERE p1.name = 'Lee' AND p2.name = 'Tom'
```

PGQL uses MATCH SHORTEST to specify shortest path search. Also, each path result is treated as a "horizontal group" and aggregates can be used to project the path.

Agenda

1 Graph Query Languages

2 Essentials for SPARQL Query & Update

3 Named Graphs for Edge Properties

4 Comparison with PG Query Languages

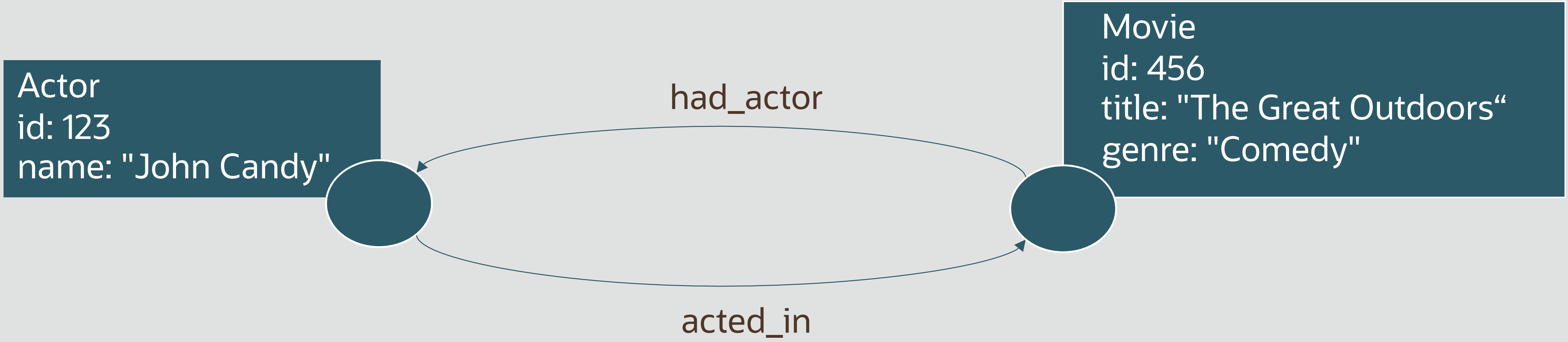
5 Graph Analytics with RDF Data

Graph Analytics with RDF Data

- RDF data model is well suited for data integration
  - Flexible data model – tolerant of dirty data
  - Semantics for merging graphs is well-defined
    - URIs
    - OWL/RDFS entailment
- We can easily extract subgraphs for analysis with graph analytics engines

Graph Analytics with RDF Data

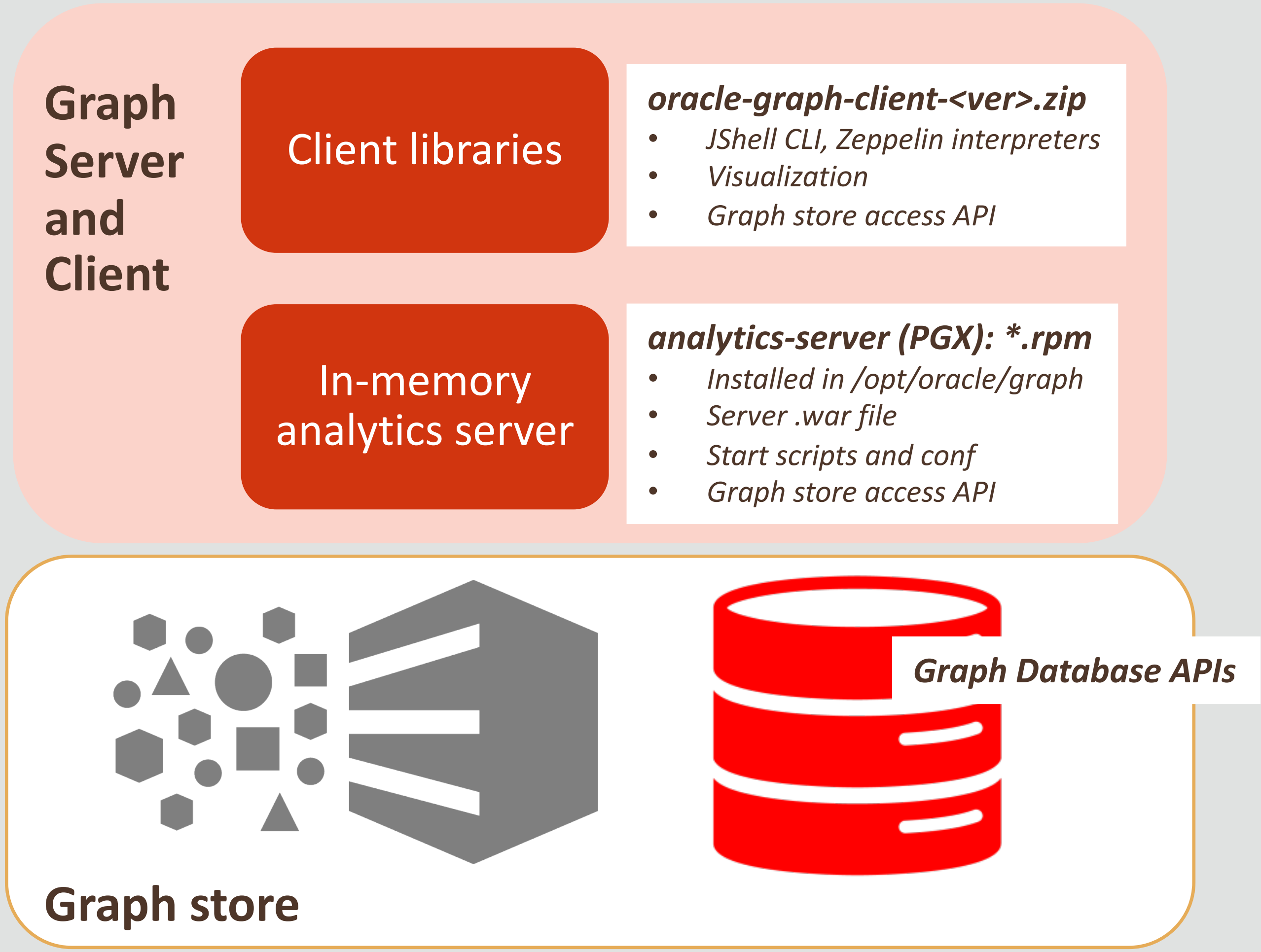
Movie/Actor Property Graph extracted from LMDB RDF Graph



Actor
id: 123
name: "John Candy"

Movie
id: 456
title: "The Great Outdoors“
genre: "Comedy"

Oracle Graph Server and Client



oracle-graph-client-<ver>.zip

- *JShell CLI, Zeppelin interpreters*
- *Visualization*
- *Graph store access API*

analytics-server (PGX): *.rpm

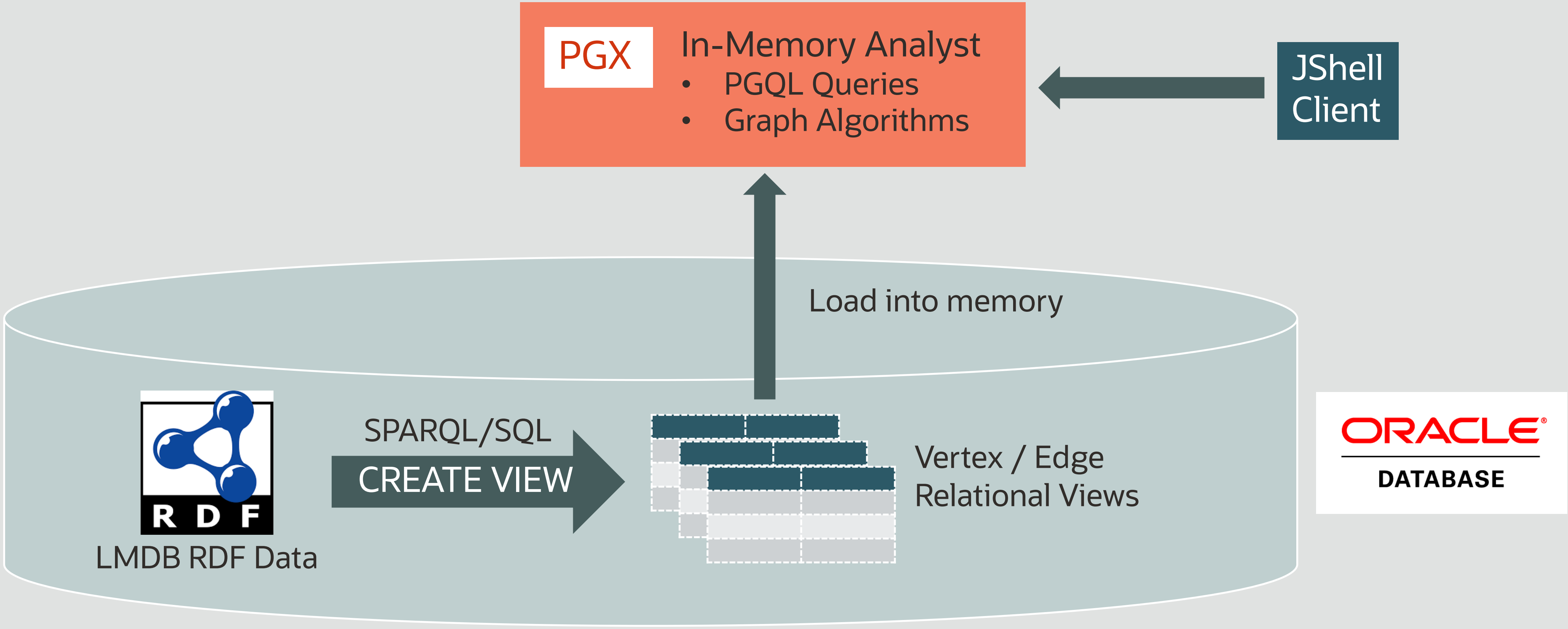
- *Installed in /opt/oracle/graph*
- *Server .war file*
- *Start scripts and conf*
- *Graph store access API*

Graph Database APIs

Software package for use with Oracle Database

- Client Libraries for building Property Graph Applications in database or in-memory
- JShell CLI, Zepplin Interpreters, Viz Application
- PGX In-memory Analytics Server
  - Run PGQL queries
  - 50 Pre-built Graph Algorithms

Workflow for Graph Analytics with RDF



PGX

In-Memory Analyst

- PGQL Queries
- Graph Algorithms

JShell Client

Load into memory

SPARQL/SQL

CREATE VIEW

Vertex / Edge Relational Views

LMDB RDF Data

Extracting the Property Graph

Create a Vertex view for Actors

```
CREATE VIEW ACTORS AS
SELECT ACTOR$RDFVID AS ID, 'Actor' AS "label", NAME AS "name"
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     PREFIX movie: <http://data.linkedmdb.org/movie/>
     SELECT ?actor ?name
     WHERE {
           ?actor rdf:type movie:actor ;
                  movie:actor_name ?name
      }',
    SEM_MODELS('LMDB'),...));
```

Extracting the Property Graph

Create a Vertex view for Movies

```
CREATE VIEW MOVIES AS
SELECT MOVIE$RDFVID AS ID, 'Movie' AS "label", MTITLE AS "title", MGENRE AS "genre"
FROM TABLE(SEM_MATCH(
   'PREFIX        rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX dcterms: <http://purl.org/dc/terms/title>
    PREFIX    movie: <http://data.linkedmdb.org/movie/>
    SELECT ?movie (MAX(STR(?genre)) AS ?mGenre) (MAX(STR(?title)) AS ?mTitle)
    WHERE {
         ?movie rdf:type movie:film ;
                    dcterms:title ?title ;
                    movie:genre/movie:film_genre_name ?genre .
    }
    GROUP BY ?movie',
   SEM_MODELS('LMDB'), ...));
```

Extracting the Property Graph

Create an Edge view for Actor-[:acted_in]->Movie

```
CREATE VIEW ACTED_IN AS
SELECT ACTOR$RDFVID AS SOURCE_ID, MOVIE$RDFVID AS DEST_ID, 'acted_in' AS "label"
FROM TABLE(SEM_MATCH(
   'PREFIX         rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX     movie: <http://data.linkedmdb.org/movie/>
    SELECT ?actor ?movie
    WHERE {
         ?movie movie:actor ?actor
    }',
   SEM_MODELS('LMDB'),…));
```

Extracting the Property Graph

Create an Edge view for Movie-[:hadActor]->Actor

```
CREATE VIEW HAD_ACTOR AS
SELECT MOVIE$RDFVID AS SOURCE_ID, ACTOR$RDFVID AS DEST_ID, 'had_actor' AS "label"
FROM TABLE(SEM_MATCH(
   'PREFIX       rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX    movie: <http://data.linkedmdb.org/movie/>
    SELECT ?actor ?movie
    WHERE {
         ?movie movie:actor ?actor
    }',
   SEM_MODELS('LMDB'),...));
```

PGX Configuration File for loading the Graph

```
{
   "name":"Imdb",
   "jdbc_url":"jdbc:oracle:thin:@localhost:1521/orcl",
   "username":"rdfuser",
   "keystore_alias":"database1",
   "vertex_id_strategy": "keys_as_ids",
   "vertex_providers":[
      {
         "name":"Actor",
         "format":"rdbms",
         "database_table_name":"ACTORS",
         "key_column":"ID",
         "key_type": "long",
         "props":[
            {
               "name":"name",
               "type":"string"
            }
         ]
      },
      {
         "name":"Movie",
         "format":"rdbms",
         "database_table_name":"MOVIES",
         "key_column":"ID",
         "key_type": "long",
         "props":[
            {
               "name":"title",
               "type":"string"
            },
            {
               "name":"genre",
               "type":"string"
            }
         ]
      }
   ],
   "edge_providers":[
      {
         "name":"acted_in",
         "format":"rdbms",
         "database_table_name":"ACTED_IN",
         "source_column":"SOURCE_ID",
         "destination_column":"DEST_ID",
         "source_vertex_provider":"Actor",
         "destination_vertex_provider":"Movie"
      },
      {
         "name":"had_actor",
         "format":"rdbms",
         "database_table_name":"HAD_ACTOR",
         "source_column":"SOURCE_ID",
         "destination_column":"DEST_ID",
         "source_vertex_provider":"Movie",
         "destination_vertex_provider":"Actor"
      }
   ]
}
```

Example PGQL Queries

Who acted in Home Alone?

```
SELECT a.name AS name
MATCH (m:Movie)-[:had_actor]->(a:Actor)
WHERE m.title = 'Home Alone'
```

Example PGQL Queries

Who are the top actors by number of movies?

```
SELECT a.name AS name, count(*) AS movieCount
MATCH (a:Actor)-[:acted_in]->(m:Movie)
GROUP BY a
ORDER BY movieCount DESC
LIMIT 10
```

Example PGQL Queries

Is there a path from Charlie Chaplin to Mr. T?

```
PATH co_star AS (:Actor)-[:acted_in]->(:Movie)<-[:acted_in]-(:Actor)
SELECT 1 AS isReachable
MATCH (a)-/:co_star+/->(b)
WHERE a.name = 'Charlie Chaplin' AND b.name = 'Mr. T'
```

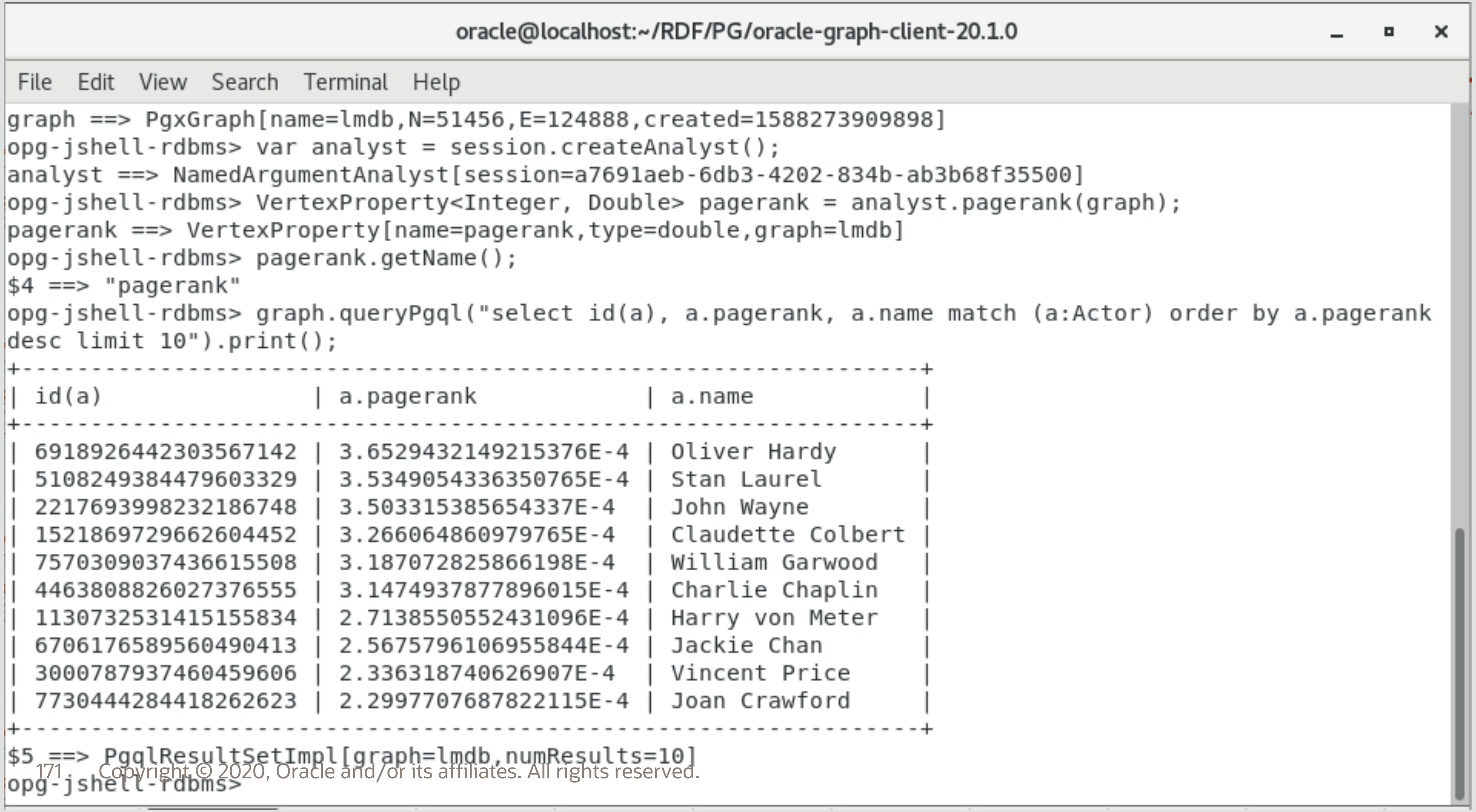
Example PGQL Queries

Find the shortest path from Charlie Chaplin to Mr. T

```
SELECT COUNT(e) AS pathLen,
            ARRAY_AGG(t.title) AS movie,
            ARRAY_AGG(t.name) AS coStar
MATCH SHORTEST ( (a) ((s)-[e:acted_in]-(t))* (b) )
WHERE a.name = 'Charlie Chaplin' AND b.name = 'Mr. T'
```

Computing Page Rank over the Graph

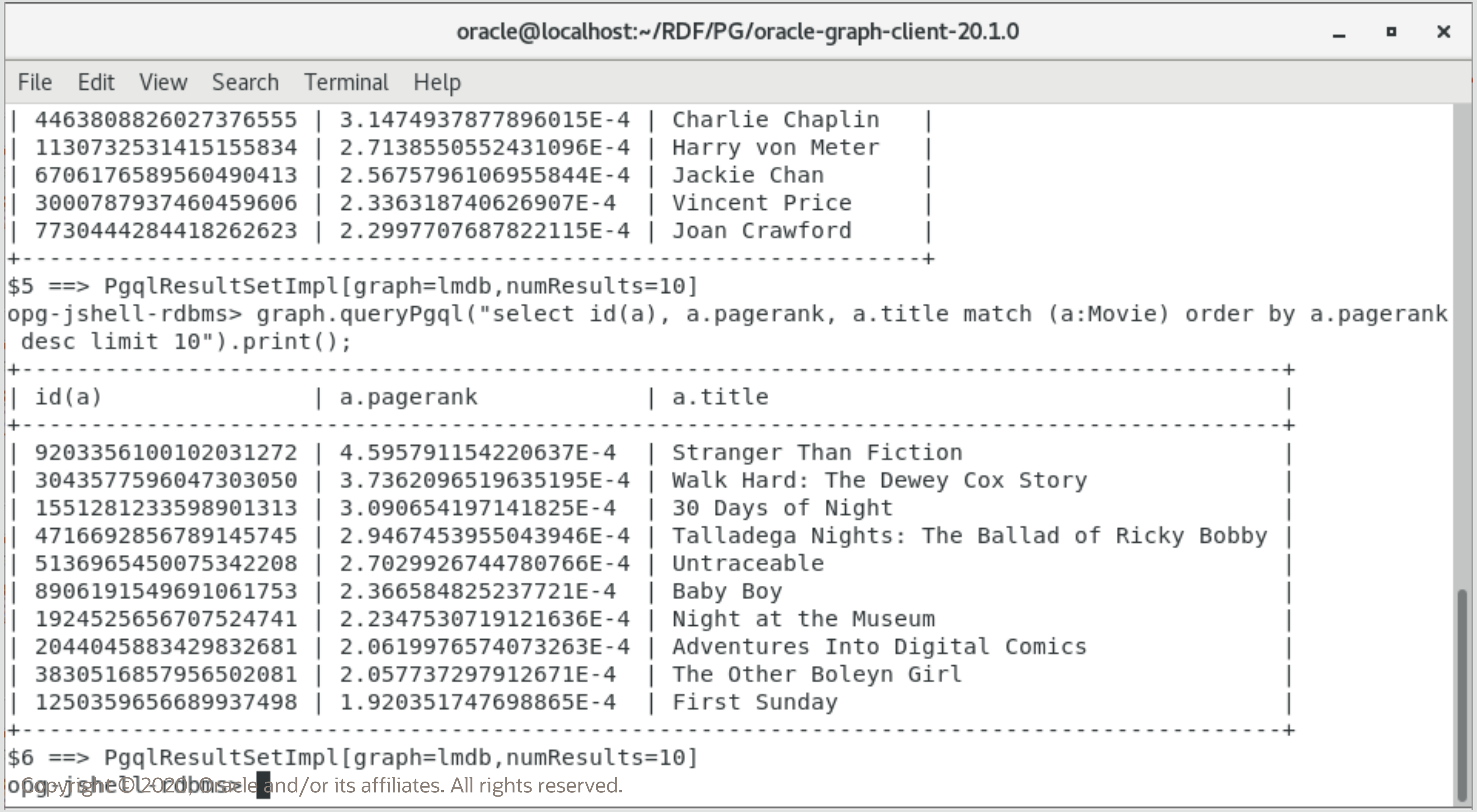
Finding the most important movies and actors


oracle@localhost:~/RDF/PG/oracle-graph-client-20.1.0
File Edit View Search Terminal Help

```
graph ==> PgxGraph[name=lmdb,N=51456,E=124888,created=1588273909898]
opg-jshell-rdbms> var analyst = session.createAnalyst();
analyst ==> NamedArgumentAnalyst[session=a7691aeb-6db3-4202-834b-ab3b68f35500]
opg-jshell-rdbms> VertexProperty<Integer, Double> pagerank = analyst.pagerank(graph);
pagerank ==> VertexProperty[name=pagerank,type=double,graph=lmdb]
opg-jshell-rdbms> pagerank.getName();
$4 ==> "pagerank"
opg-jshell-rdbms> graph.queryPgql("select id(a), a.pagerank, a.name match (a:Actor) order by a.pagerank
desc limit 10").print();
+-------------------------------------------------------------------+
| id(a)                | a.pagerank             | a.name            |
+-------------------------------------------------------------------+
| 6918926442303567142 | 3.6529432149215376E-4 | Oliver Hardy       |
| 5108249384479603329 | 3.5349054336350765E-4 | Stan Laurel        |
| 2217693998232186748 | 3.503315385654337E-4  | John Wayne         |
| 1521869729662604452 | 3.266064860979765E-4  | Claudette Colbert |
| 7570309037436615508 | 3.187072825866198E-4  | William Garwood    |
| 4463808826027376555 | 3.1474937877896015E-4 | Charlie Chaplin    |
| 1130732531415155834 | 2.7138550552431096E-4 | Harry von Meter    |
| 6706176589560490413 | 2.5675796106955844E-4 | Jackie Chan        |
| 3000787937460459606 | 2.336318740626907E-4  | Vincent Price      |
| 7730444284418262623 | 2.2997707687822115E-4 | Joan Crawford      |
+-------------------------------------------------------------------+
$5 ==> PgqlResultSetImpl[graph=lmdb,numResults=10]
opg-jshell-rdbms>
```

Computing Page Rank over the Graph

Finding the most important movies and actors


oracle@localhost:~/RDF/PG/oracle-graph-client-20.1.0
File Edit View Search Terminal Help

```
| 4463808826027376555 | 3.147493787789601 5E-4 | Charlie Chaplin    |
| 1130732531415155834 | 2.713855055243109 6E-4 | Harry von Meter    |
| 6706176589560490413 | 2.567579610695584 4E-4 | Jackie Chan        |
| 3000787937460459606 | 2.336318740626907E-4   | Vincent Price      |
| 7730444284418262623 | 2.299770768782211 5E-4 | Joan Crawford      |
+-------------------------------------------------------------------+
$5 ==> PgqlResultSetImpl[graph=lmdb,numResults=10]
opg-jshell-rdbms> graph.queryPgql("select id(a), a.pagerank, a.title match (a:Movie) order by a.pagerank
 desc limit 10").print();
+----------------------------------------------------------------------------------------------+
| id(a)               | a.pagerank             | a.title                                      |
+----------------------------------------------------------------------------------------------+
| 9203356100102031272 | 4.595791154220637E-4   | Stranger Than Fiction                        |
| 3043577596047303050 | 3.7362096519635195E-4  | Walk Hard: The Dewey Cox Story               |
| 1551281233598901313 | 3.090654197141825E-4   | 30 Days of Night                             |
| 4716692856789145745 | 2.9467453955043946E-4  | Talladega Nights: The Ballad of Ricky Bobby  |
| 5136965450075342208 | 2.7029926744780766E-4  | Untraceable                                  |
| 8906191549691061753 | 2.366584825237721E-4   | Baby Boy                                     |
| 1924525656707524741 | 2.2347530719121636E-4  | Night at the Museum                          |
| 2044045883429832681 | 2.0619976574073263E-4  | Adventures Into Digital Comics               |
| 3830516857956502081 | 2.057737297912671E-4   | The Other Boleyn Girl                        |
| 1250359656689937498 | 1.920351747698865E-4   | First Sunday                                 |
+----------------------------------------------------------------------------------------------+
$6 ==> PgqlResultSetImpl[graph=lmdb,numResults=10]
opg-jshell-rdbms>
```

Agenda

Part 1

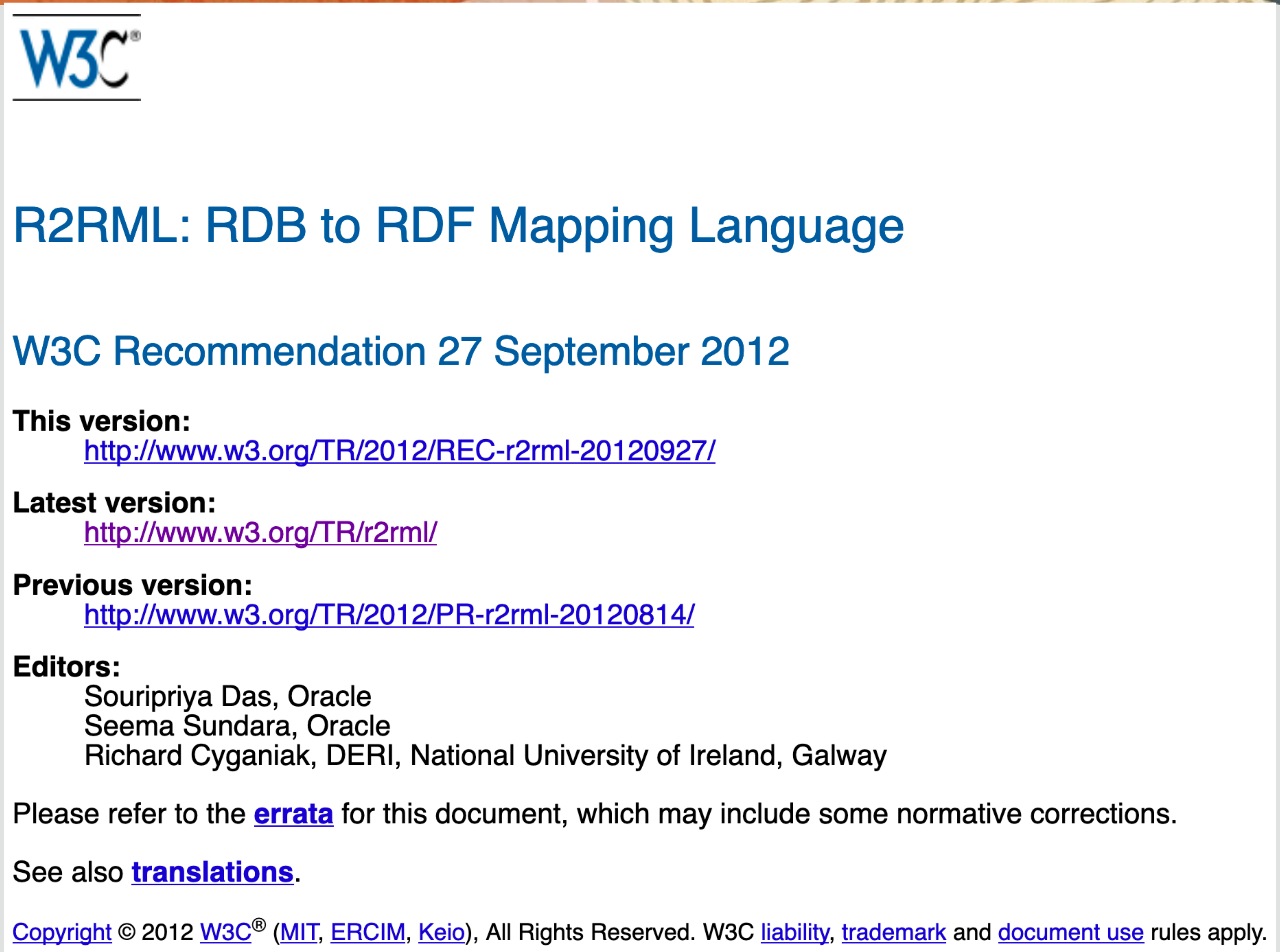
- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo



R2RML: RDB to RDF Mapping Language

W3C Recommendation 27 September 2012

This version:
http://www.w3.org/TR/2012/REC-r2rml-20120927/

Latest version:
http://www.w3.org/TR/r2rml/

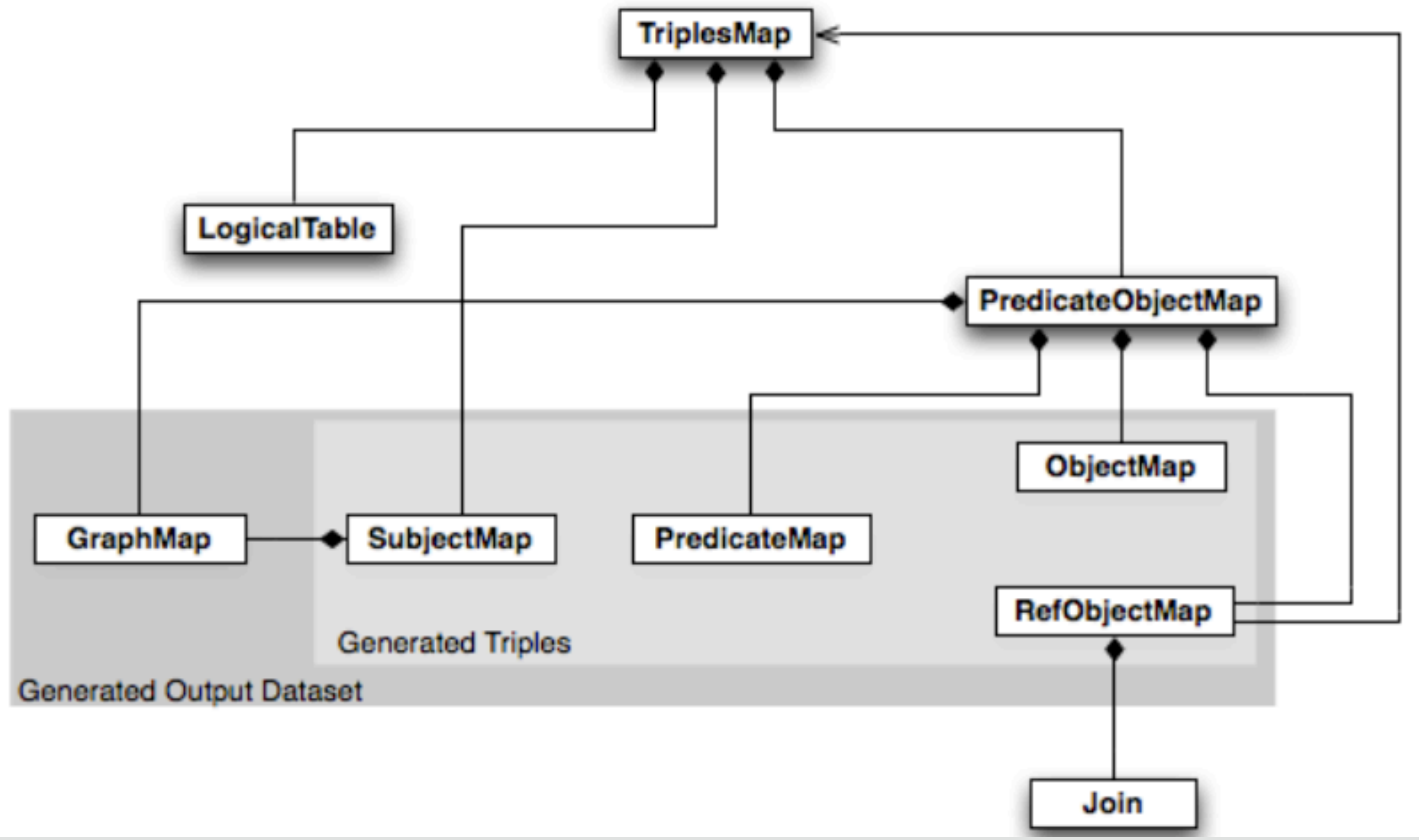
Previous version:
http://www.w3.org/TR/2012/PR-r2rml-20120814/

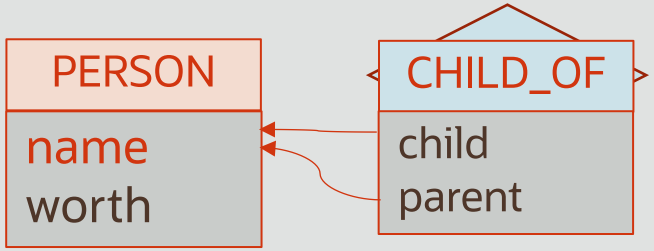
Editors:
Souripriya Das, Oracle
Seema Sundara, Oracle
Richard Cyganiak, DERI, National University of Ireland, Galway

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

See also **translations**.

R2RML map. doc = {TriplesMap$_1$, TriplesMap$_2$, …}



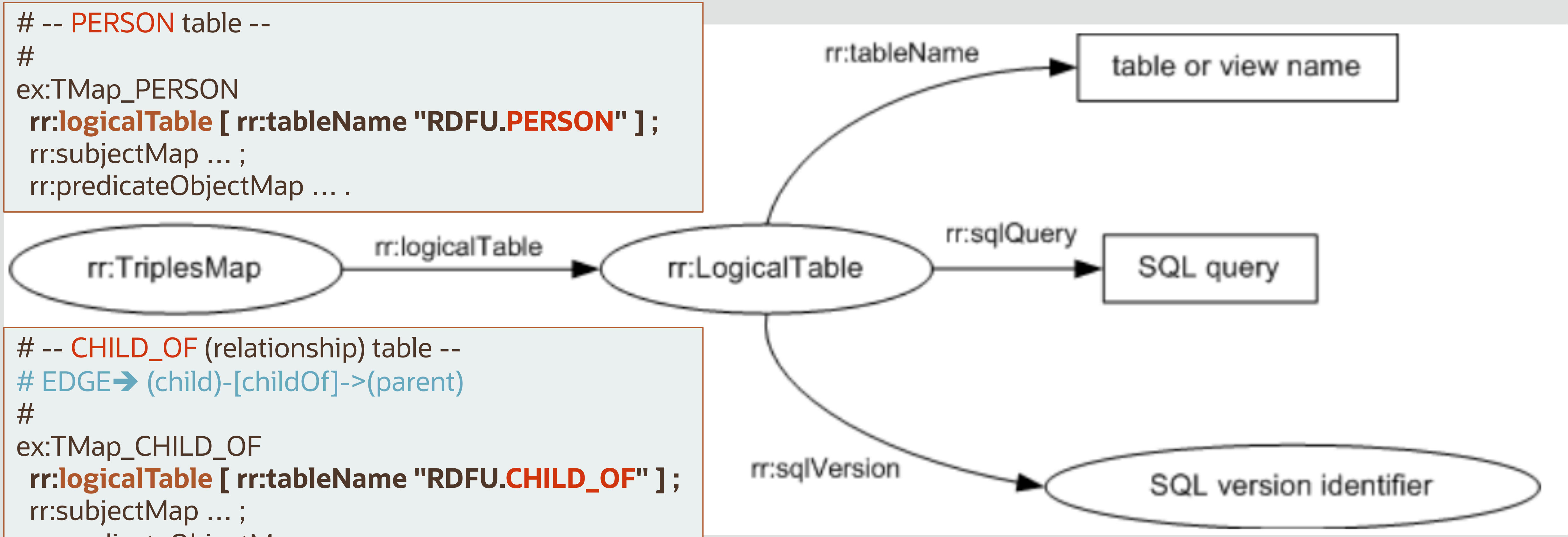


```
# -- PERSON table --
#
ex:TMap_PERSON  a rr:TriplesMap ;
  rr:logicalTable … ;
  rr:subjectMap … ;
  rr:predicateObjectMap … .
```

```
# -- CHILD_OF (relationship) table --
# EDGE➔ (child)-[childOf]->(parent)
#
ex:TMap_CHILD_OF a rr:TriplesMap ;
  rr:logicalTable … ;
  rr:subjectMap … ;
  rr:predicateObjectMap … .
```

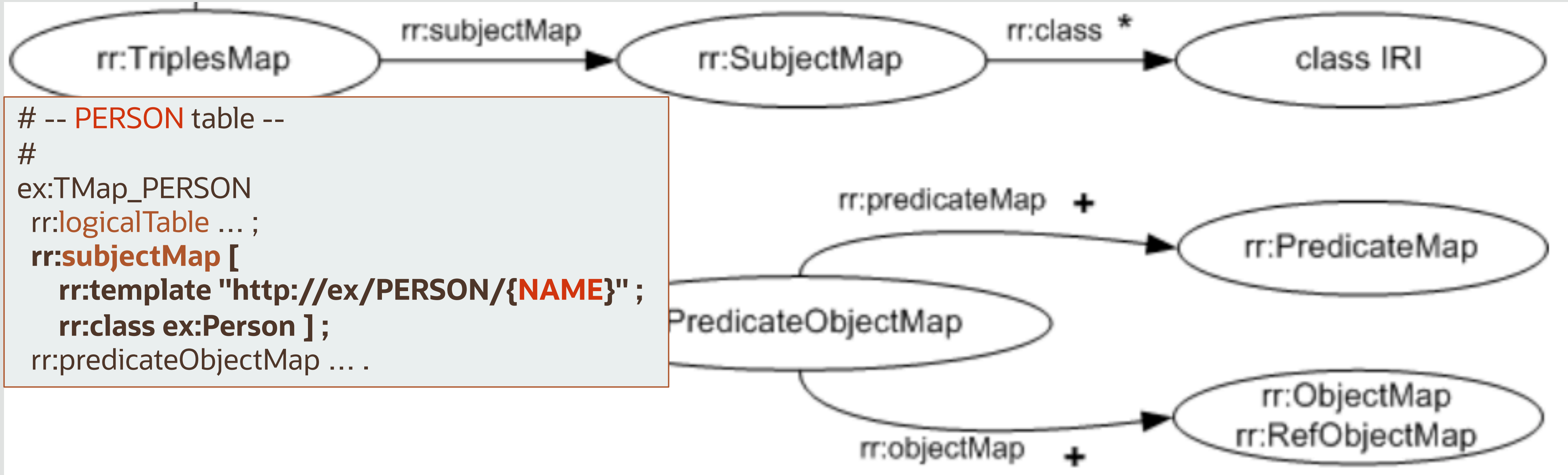
R2RML: TriplesMap $\rightarrow^{1}$ LogicalTable

```
# -- PERSON table --
#
ex:TMap_PERSON
  rr:logicalTable [ rr:tableName "RDFU.PERSON" ] ;
  rr:subjectMap … ;
  rr:predicateObjectMap … .
```



```
# -- CHILD_OF (relationship) table --
# EDGE➔ (child)-[childOf]->(parent)
#
ex:TMap_CHILD_OF
  rr:logicalTable [ rr:tableName "RDFU.CHILD_OF" ] ;
  rr:subjectMap … ;
  rr:predicateObjectMap … .
```

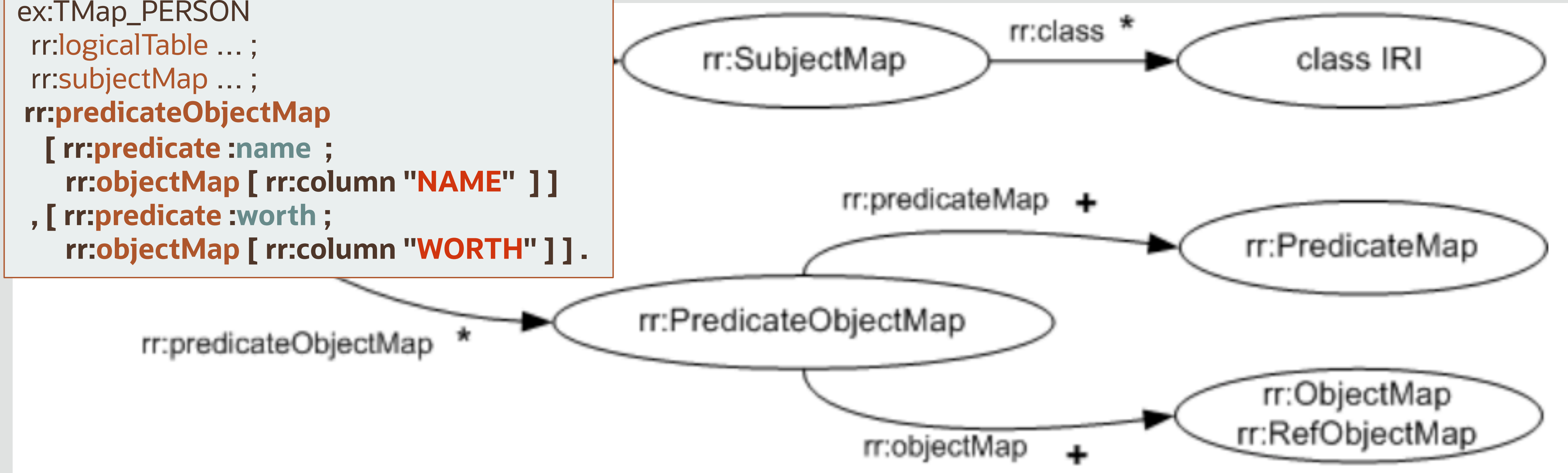
R2RML: TriplesMap $\to^{1}$ SubjectMap, $\to^{*}$ PredicateObjectMap



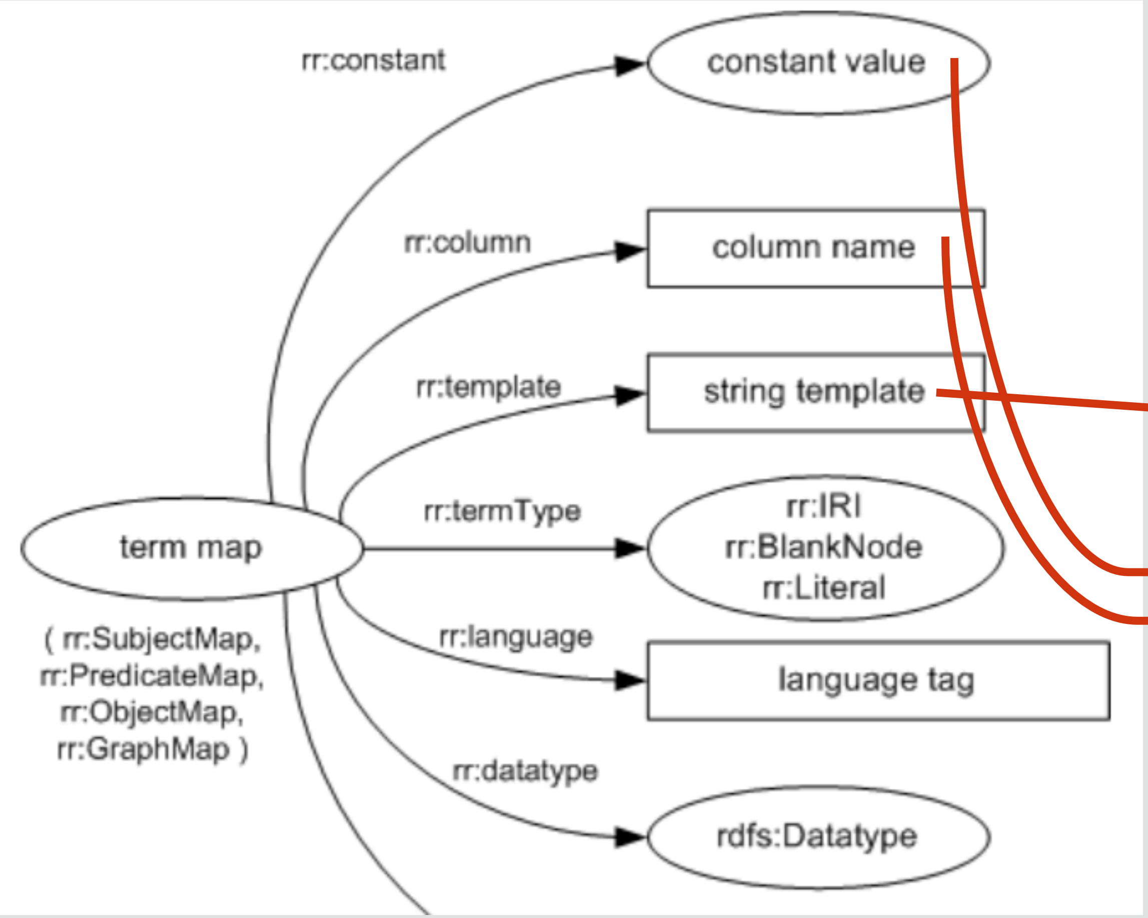
```
# -- PERSON table --
#
ex:TMap_PERSON
 rr:logicalTable … ;
 rr:subjectMap [
   rr:template "http://ex/PERSON/{NAME}" ;
   rr:class ex:Person ] ;
 rr:predicateObjectMap … .
```

R2RML: TriplesMap $\rightarrow^{1}$ SubjectMap, $\rightarrow^{*}$ PredicateObjectMap

```
# -- PERSON table --
#
ex:TMap_PERSON
  rr:logicalTable … ;
  rr:subjectMap … ;
 rr:predicateObjectMap
   [ rr:predicate :name  ;
     rr:objectMap [ rr:column "NAME"  ] ]
 , [ rr:predicate :worth ;
     rr:objectMap [ rr:column "WORTH" ] ] .
```



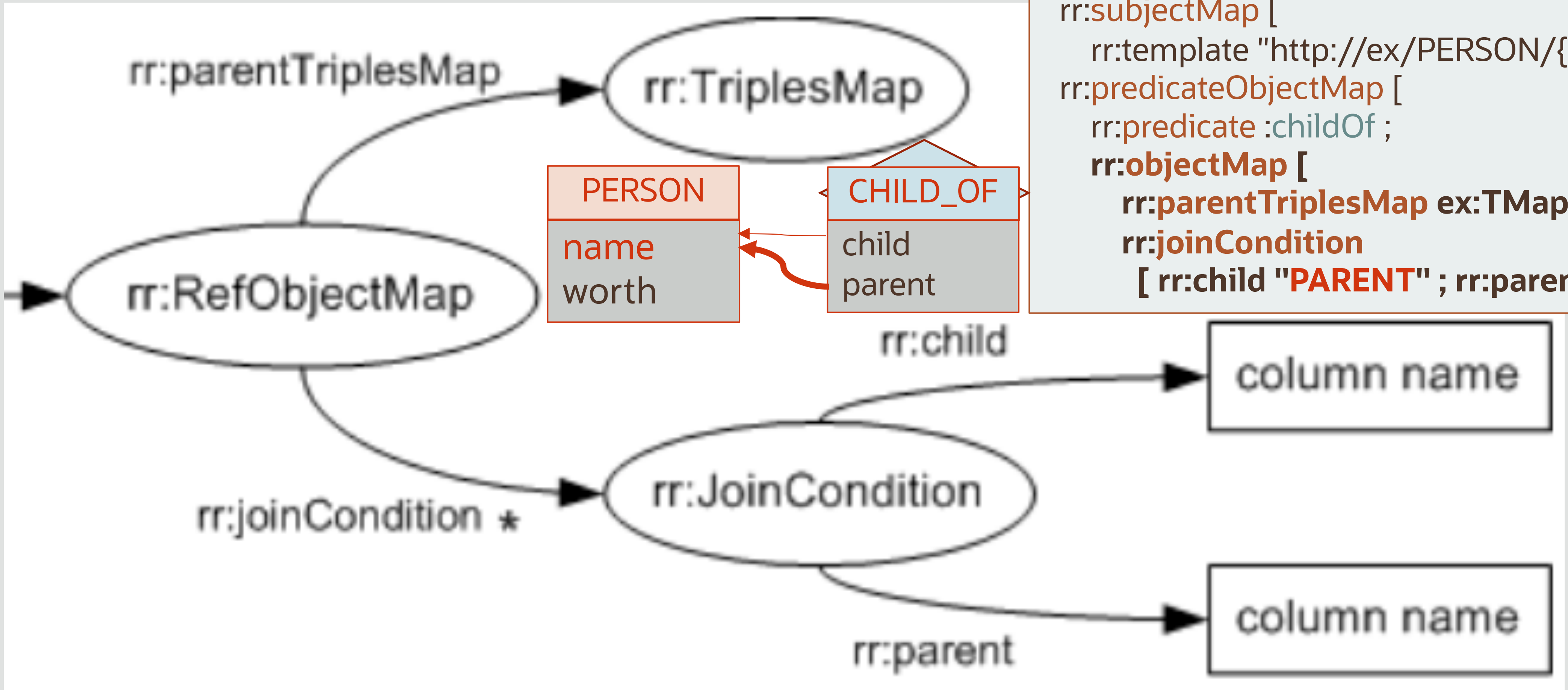
R2RML: SubjectMap, PredicateMap, ObjectMap



```
# -- PERSON table --
#
ex:TMap_PERSON
  rr:logicalTable ... ;
 rr:subjectMap [
    rr:template "http://ex/PERSON/{NAME}" ;
    rr:class ex:Person ] ;
 rr:predicateObjectMap
  [ rr:predicate :name  ;
    rr:objectMap [ rr:column "NAME"  ] ]
  , [ rr:predicate :worth ;
    rr:objectMap [ rr:column "WORTH" ] ] .
```

R2RML: RefObjectMap

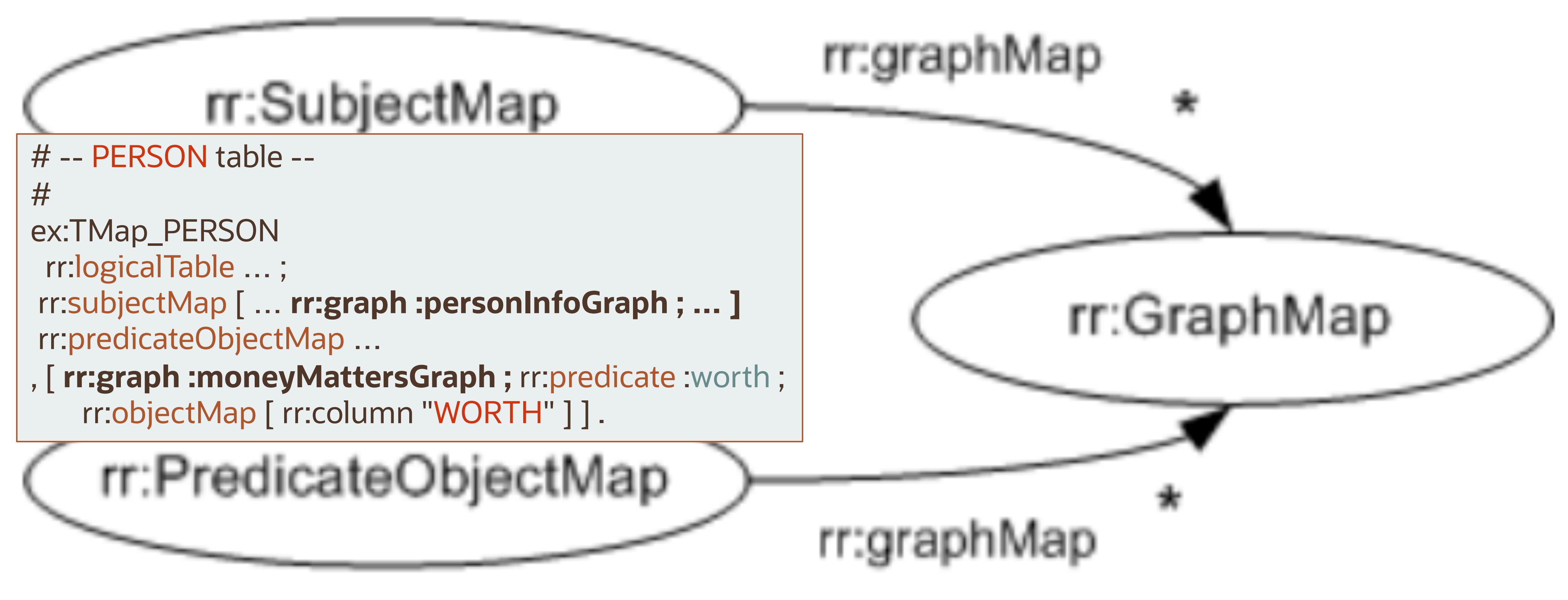
```
# -- CHILD_OF (relationship) table --
# EDGE➔ (child)-[childOf]->(parent)
#
ex:TMap_CHILD_OF
  rr:logicalTable … ;
  rr:subjectMap [
     rr:template "http://ex/PERSON/{CHILD}" … ] ;
  rr:predicateObjectMap [
     rr:predicate :childOf ;
     rr:objectMap [
        rr:parentTriplesMap ex:TMap_PERSON ;
        rr:joinCondition
          [ rr:child "PARENT" ; rr:parent "NAME" ] ] ] .
```



PERSON
name
worth

CHILD_OF
child
parent

R2RML: GraphMap



```
# -- PERSON table --
#
ex:TMap_PERSON
  rr:logicalTable … ;
 rr:subjectMap [ … rr:graph :personInfoGraph ; … ]
 rr:predicateObjectMap …
, [ rr:graph :moneyMattersGraph ; rr:predicate :worth ;
      rr:objectMap [ rr:column "WORTH" ] ] .
```

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Resources for Getting Started

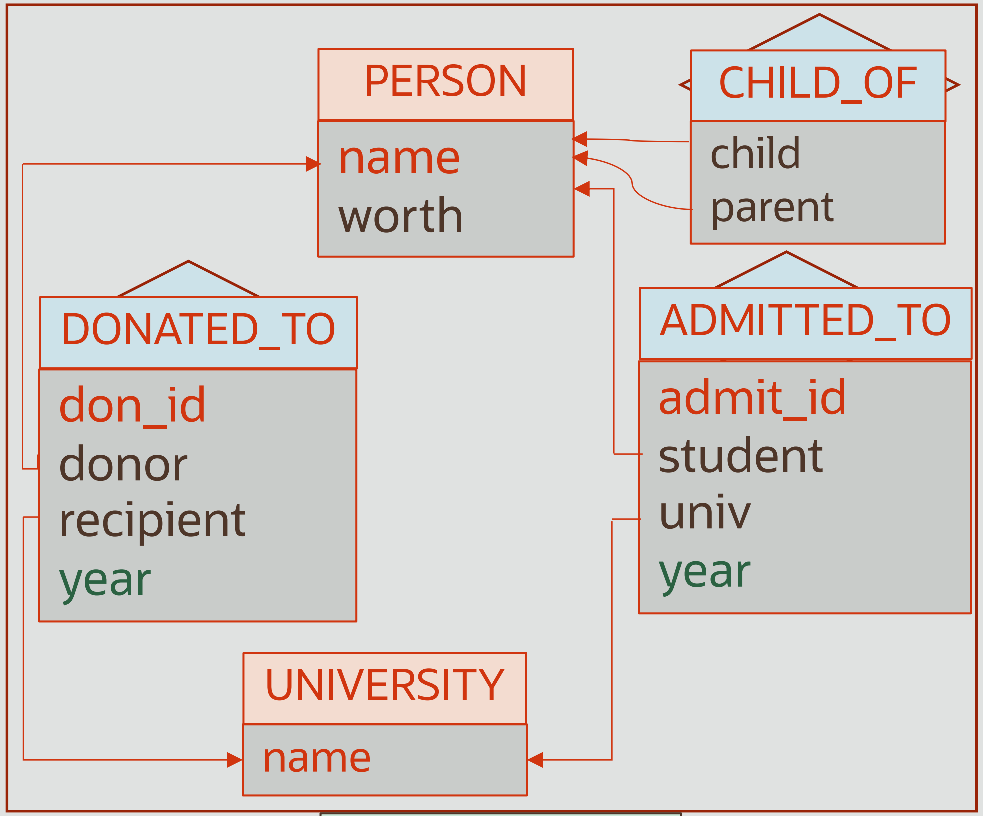
- VM image: : https://www.oracle.com/database/technologies/databaseappdev-vm.html

- Oracle Database Docker
  Single instance database from
  https://github.com/oracle/docker-images/tree/master/OracleDatabase

- Oracle Cloud
  Use **Oracle Database Cloud Service** with $300 free credits
  On the roadmap: RDF Graph support in ‘Always Free Tier’

Relational to RDF Quads:
Example: ER model and Relational Data

John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.
Mary, a child of John, got admitted to Top University in 2011.


PERSON
name
worth

CHILD_OF
child
parent

DONATED_TO
don_id
donor
recipient
year

ADMITTED_TO
admit_id
student
univ
year

UNIVERSITY
name

ER diagram

PERSON

name	worth
John	1 Bil
Mary	

CHILD_OF

child	parent
Mary	John

DONATED_TO

don_id	donor	recipient	year
1	John	TopUniv	2010
2	John	TopUniv	2012

UNIVERSITY

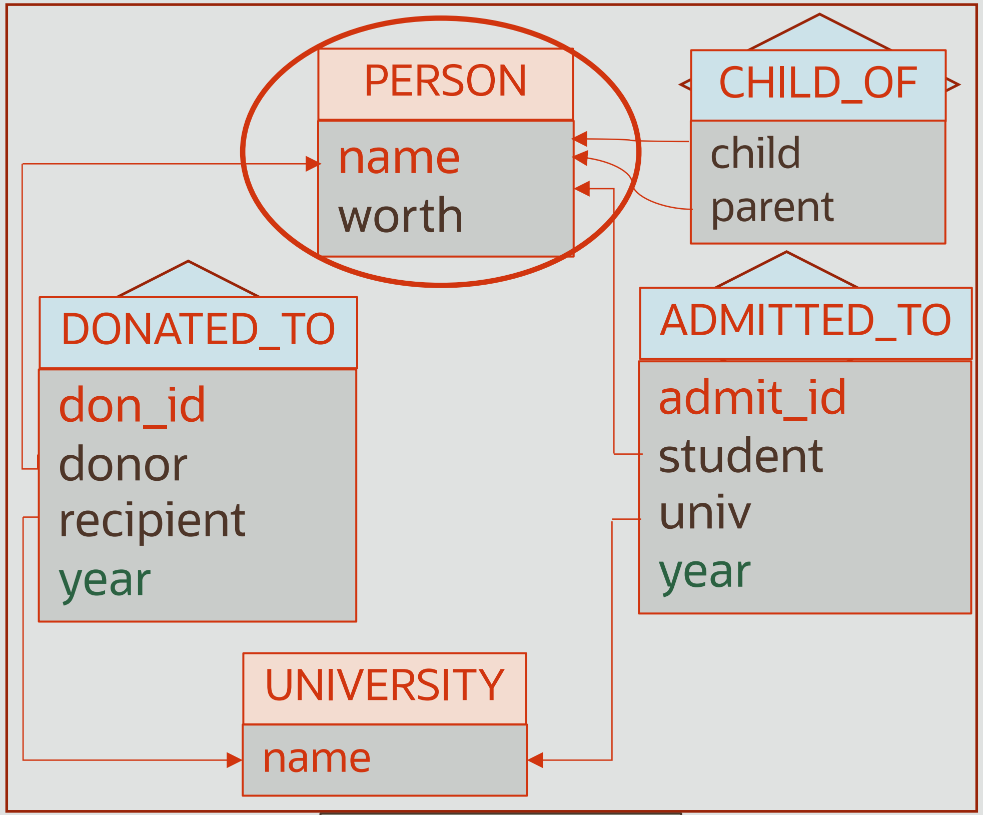
name
TopUniv

ADMITTED_TO

admit_id	student	univ	year
1	Mary	TopUniv	2011

Relational Data

Relational to RDF Quads: Entity in ER model ➔ R2RML

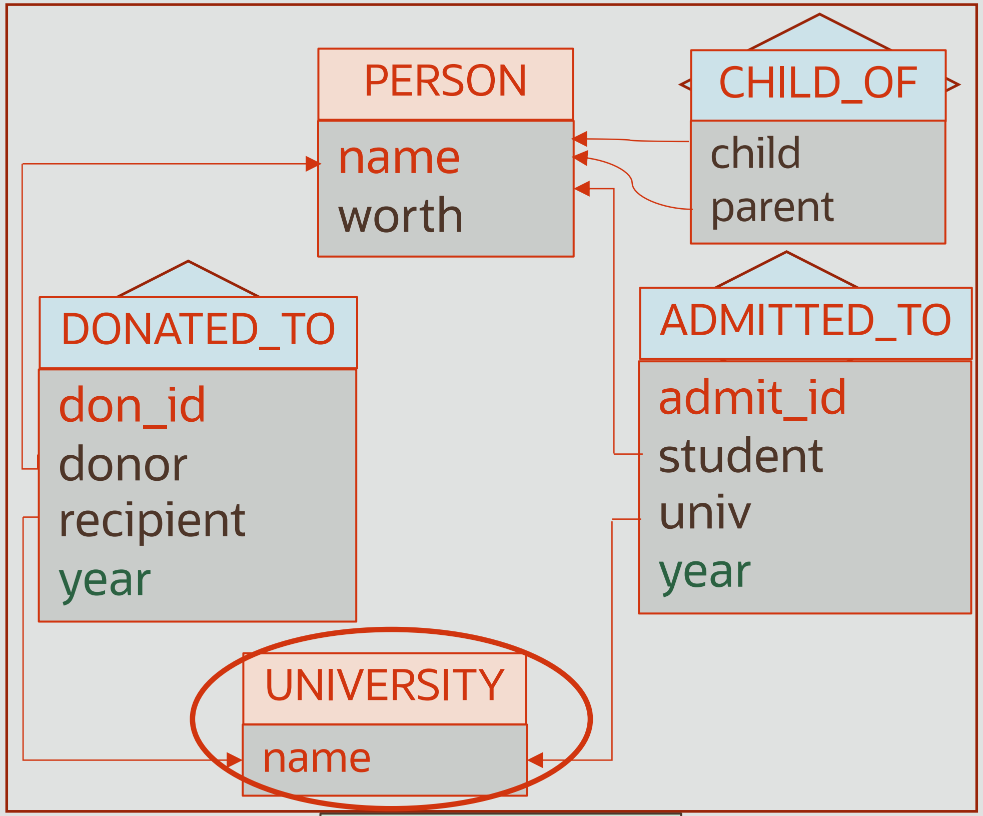


ER diagram

```
# -- PERSON table --
#
ex:TMap_PERSON
  rr:logicalTable [ rr:tableName "RDFU.PERSON" ] ;

  rr:subjectMap [
     rr:template "http://ex/PERSON/{NAME}" ;
     rr:class ex:Person ] ;

  rr:predicateObjectMap
   [ rr:predicate :name  ; rr:objectMap [ rr:column "NAME"  ] ]
  , [ rr:predicate :worth ; rr:objectMap [ rr:column "WORTH" ] ] .
```

R2RML mapping

Relational to RDF Quads:
Entity in ER model ➔ R2RML


PERSON
name
worth

CHILD_OF
child
parent

DONATED_TO
don_id
donor
recipient
year

ADMITTED_TO
admit_id
student
univ
year

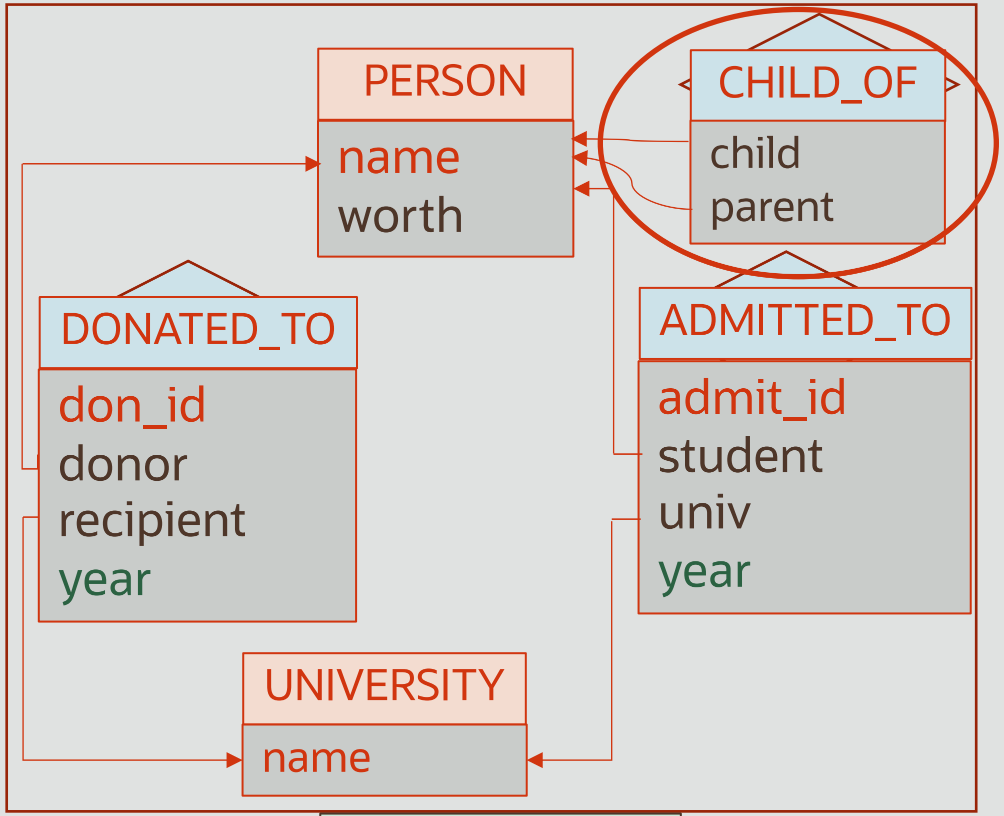
UNIVERSITY
name

ER diagram

```
# -- UNIVERSITY table --
#
ex:TMap_UNIVERSITY
  rr:logicalTable [ rr:tableName "RDFU.UNIVERSITY" ] ;

  rr:subjectMap [
      rr:template "http://ex/UNIVERSITY/{NAME}" ;
      rr:class ex:University ] ;

  rr:predicateObjectMap
    [ rr:predicate :name  ;  rr:objectMap [ rr:column "NAME"  ] ] .
```

R2RML mapping

Relational to RDF Quads:
Relation in ER model ➔ R2RML



PERSON
name
worth

CHILD_OF
child
parent

DONATED_TO
don_id
donor
recipient
year

ADMITTED_TO
admit_id
student
univ
year

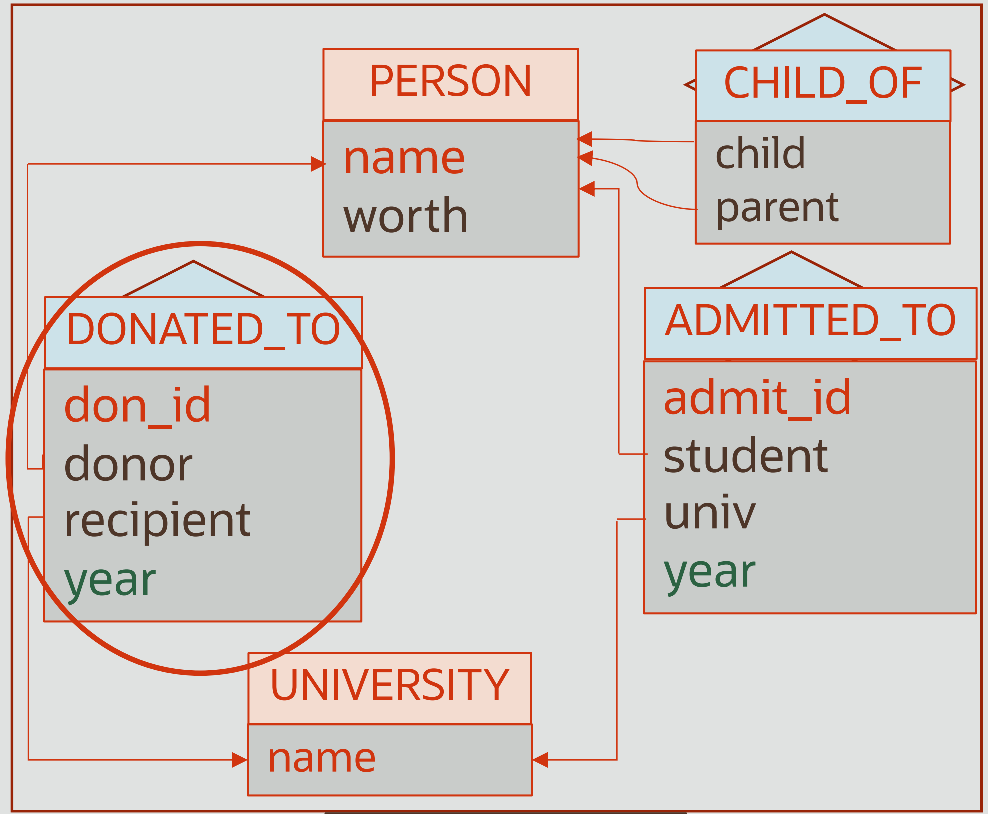
UNIVERSITY
name

ER diagram

```
# -- CHILD_OF (relationship) table --
# EDGE➔ (child)-[childOf]->(parent)
#
ex:TMap_CHILD_OF
  rr:logicalTable [ rr:tableName "RDFU.CHILD_OF" ] ;

  rr:subjectMap [
    rr:template "http://ex/PERSON/{CHILD}" ;
    rr:class ex:Child ] ;

  rr:predicateObjectMap [
    rr:predicate :childOf ;
    rr:objectMap [
      rr:parentTriplesMap ex:TMap_PERSON ;
      rr:joinCondition [ rr:child "PARENT" ; rr:parent "NAME" ] ] ] .
```

R2RML mapping

Relational to RDF Quads: Relation-As-Entity in ER → R2RML

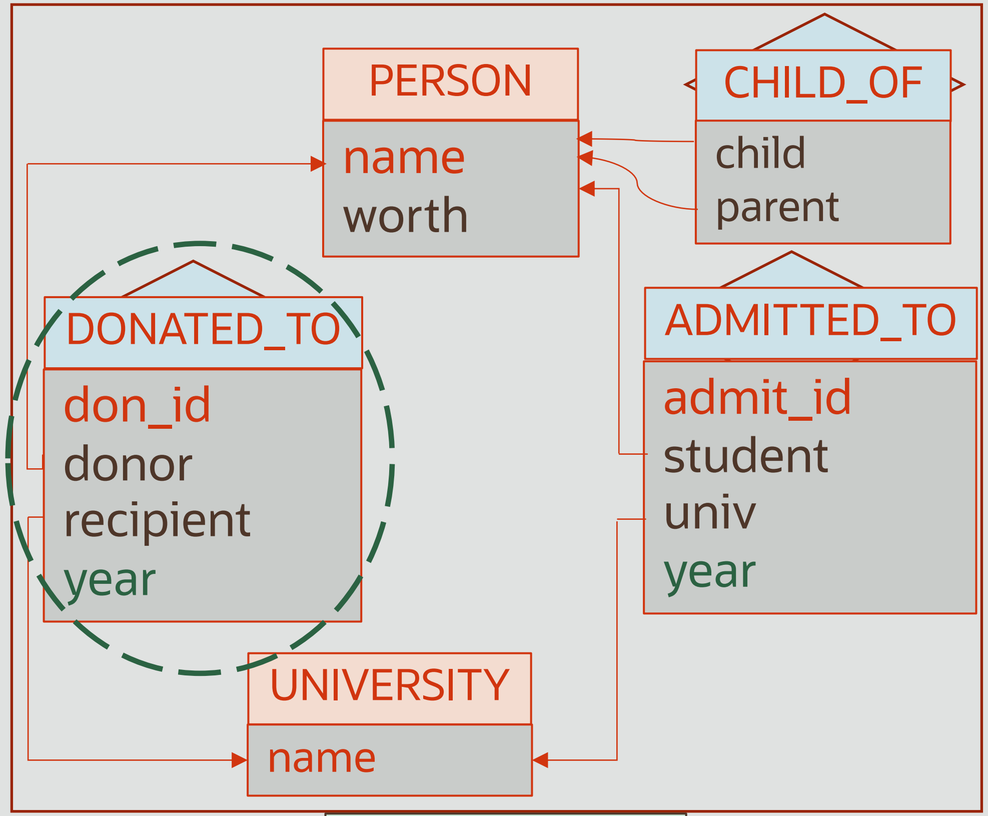


ER diagram

```
# -- DONATED_TO (relationship-as-entity) table --
#
ex:TMap_DONATED_TO_AS_ENTITY
  rr:logicalTable [ rr:tableName "RDFU.DONATED_TO" ] ;

  rr:subjectMap [
      rr:template "http://ex/donationId#{DON_ID}" ;
      rr:class ex:Donation ] ;
  …
  rr:predicateObjectMap
    [ rr:predicate :year  ; rr:objectMap [ rr:column "YEAR"  ] ] .
```

R2RML mapping

Relational to RDF Quads:
Relation in ER model ➔ R2RML

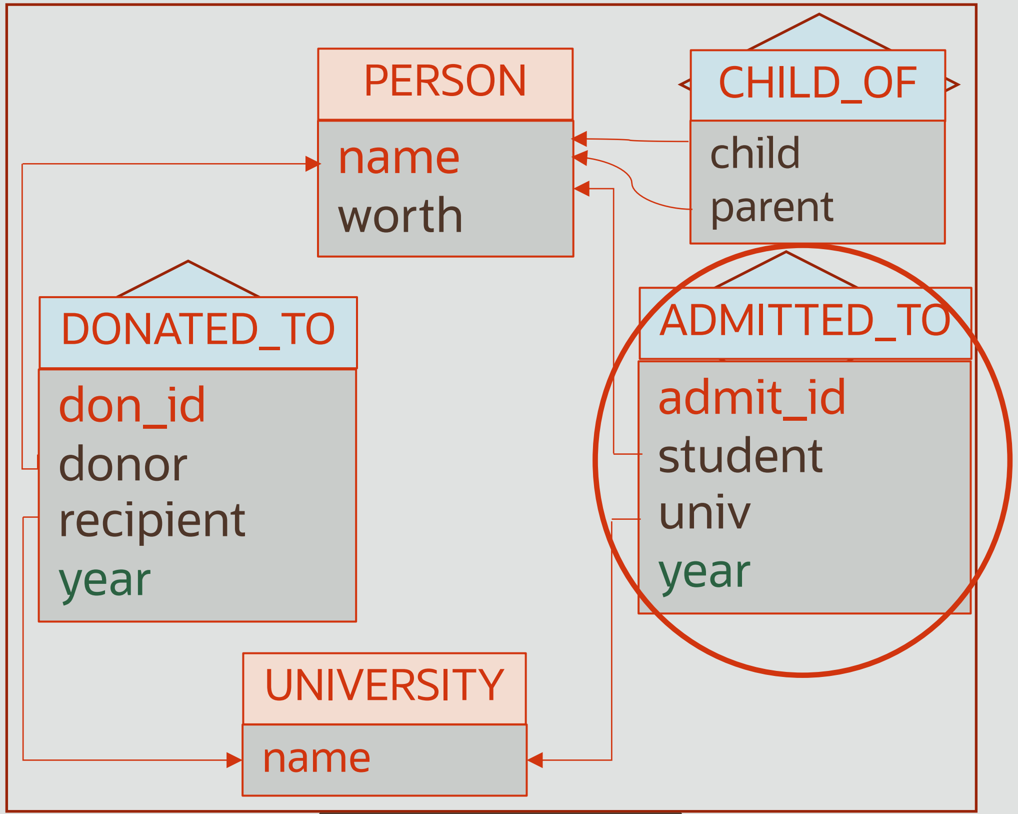


ER diagram

```
# -- DONATED_TO (relationship) table --
# EDGE➔ (donor)-[donatedTo]->(recipient)
ex:TMap_DONATED_TO
 rr:logicalTable [ rr:tableName "RDFU.DONATED_TO" ] ;

 rr:subjectMap [
   rr:template "http://ex/PERSON/{DONOR}" ;
   rr:class ex:Donor ] ;

 rr:predicateObjectMap [
   rr:graphMap [ rr:template "http://ex/donationId#{DON_ID}" ] ;
   rr:predicate :donatedTo ;
   rr:objectMap [
     rr:parentTriplesMap ex:TMap_UNIVERSITY ;
     rr:joinCondition [ rr:child "RECIPIENT" ; rr:parent "NAME" ] ] ] .
```

R2RML mapping

Relational to RDF Quads:
Relation-As-Entity in ER ➔ R2RML

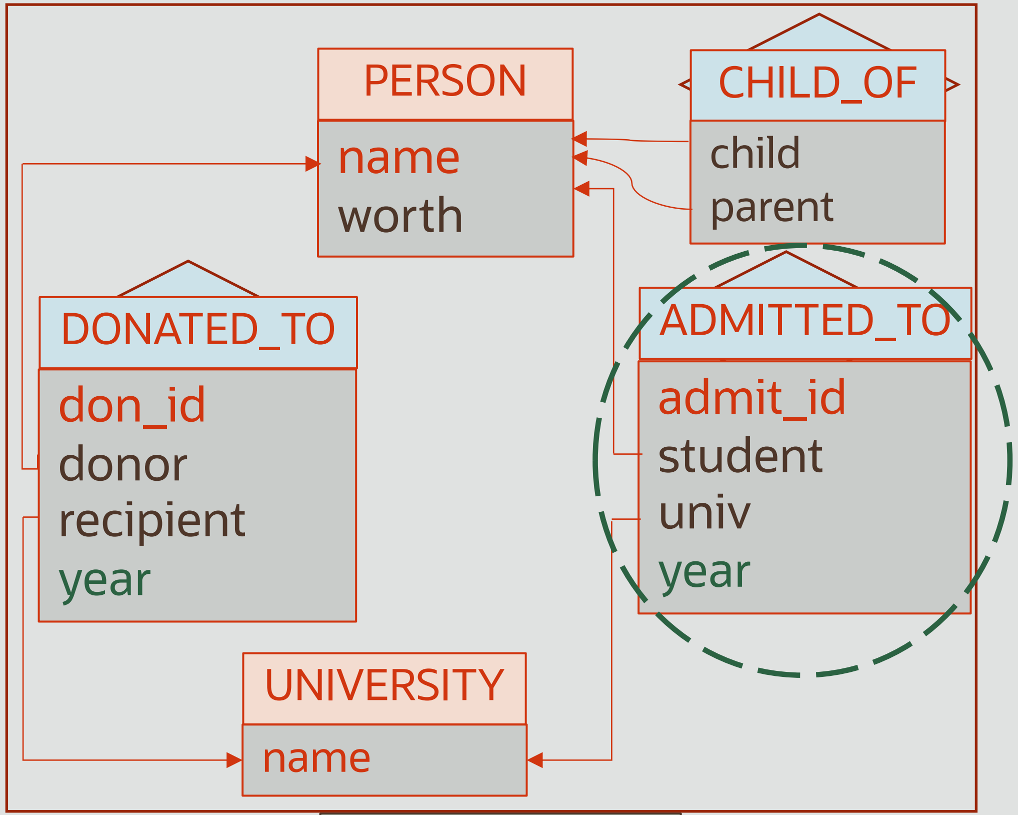


ER diagram

```
# -- ADMITTED_TO (relationship-as-entity) table --
#
ex:TMap_ADMITTED_TO_AS_ENTITY
  rr:logicalTable [ rr:tableName "RDFU.ADMITTED_TO" ] ;

  rr:subjectMap [
     rr:template "http://ex/admissionId#{ADMIT_ID}" ;
     rr:class ex:Admission ] ;
  …
  rr:predicateObjectMap
   [ rr:predicate :year  ; rr:objectMap [ rr:column "YEAR"  ] ] .
```

R2RML mapping

Relational to RDF Quads:
Relation in ER model ➔ R2RML



ER diagram

```
# -- ADMITTED_TO (relationship) table --
# EDGE➔ (student)-[admittedTo]->(univ)
ex:TMap_ADMITTED_TO
  rr:logicalTable [ rr:tableName "RDFU.ADMITTED_TO" ] ;

  rr:subjectMap [
    rr:template "http://ex/PERSON/{STUDENT}" ;
    rr:class ex:Admitted ] ;

  rr:predicateObjectMap [
    rr:graphMap [ rr:template "http://ex/admissionId#{ADMIT_ID}" ] ;
    rr:predicate :admittedTo ;
    rr:objectMap [
      rr:parentTriplesMap ex:TMap_UNIVERSITY ;
      rr:joinCondition [ rr:child "UNIV" ; rr:parent "NAME" ] ] ] .
```

R2RML mapping

Relational to RDF Quads:
Resulting RDF Graph, SPARQL query

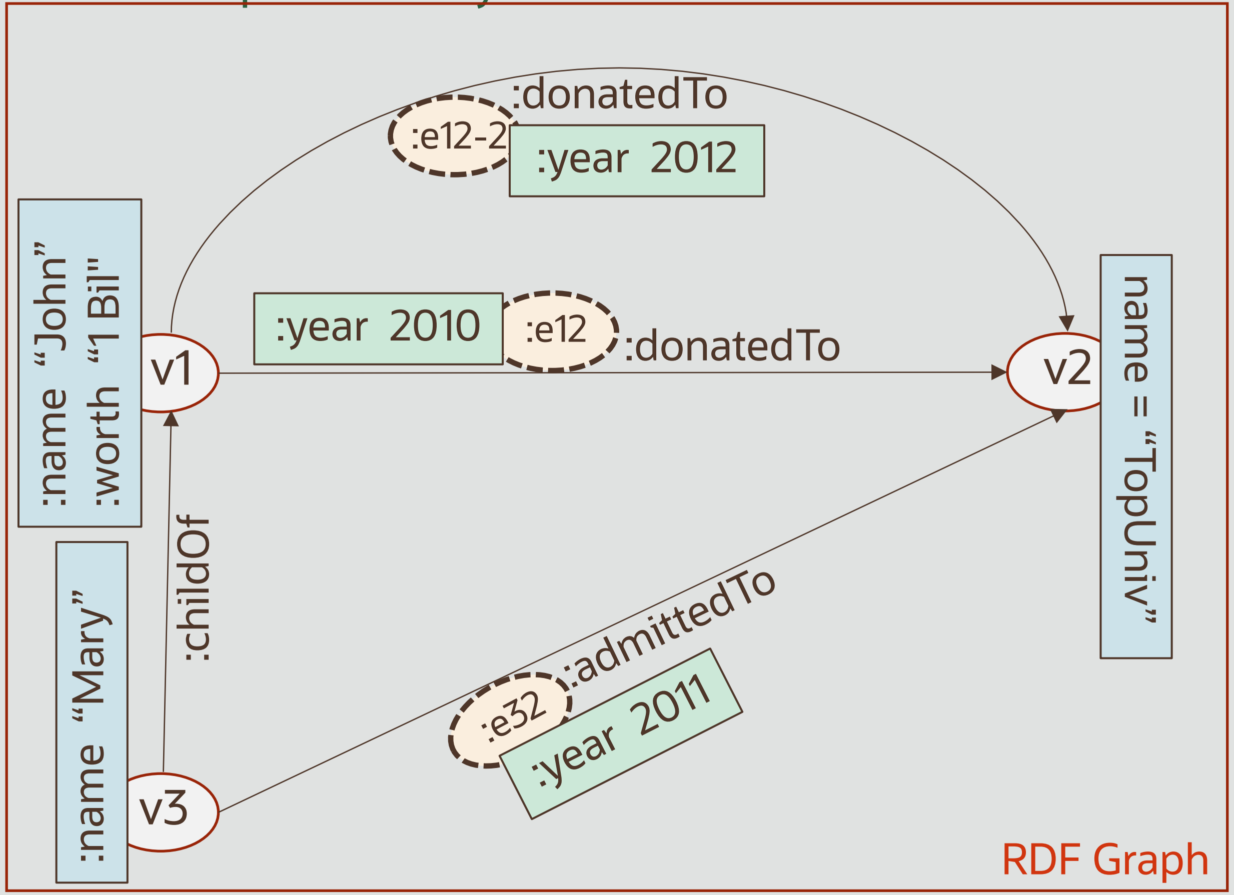
John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.
Mary, a child of John, got admitted to Top University in 2011.

Find names of parent, university, and child where parent donated to the university during a year and his/her child got admitted to the university in the following year.

```
SELECT ?paName ?univName ?chName
WHERE {
  ?child     :childOf    ?parent .
  #
  graph ?donEdge { ?parent   :donatedTo   ?univ }
  ?donEdge   :year       ?donYear .
  #
  graph ?admEdge { ?child    :admittedTo  ?univ }
  ?admEdge   :year       ?admYear .
  #
  FILTER ( ?admYear = ?donYear + 1 )
  ?child     :name       ?chName .
  ?parent    :name       ?paName .
  ?univ      :name       ?univName }
```

triple name is specified as graph name.

SPARQL Query



RDF Graph

:e12 ➔<http://ex/donationId#1>, :e12-2 ➔<http://ex/donationId#2>,
:e32 ➔<http://ex/admissionId#1>

Relational to RDF Quads:

Schema for Resulting RDF Graph

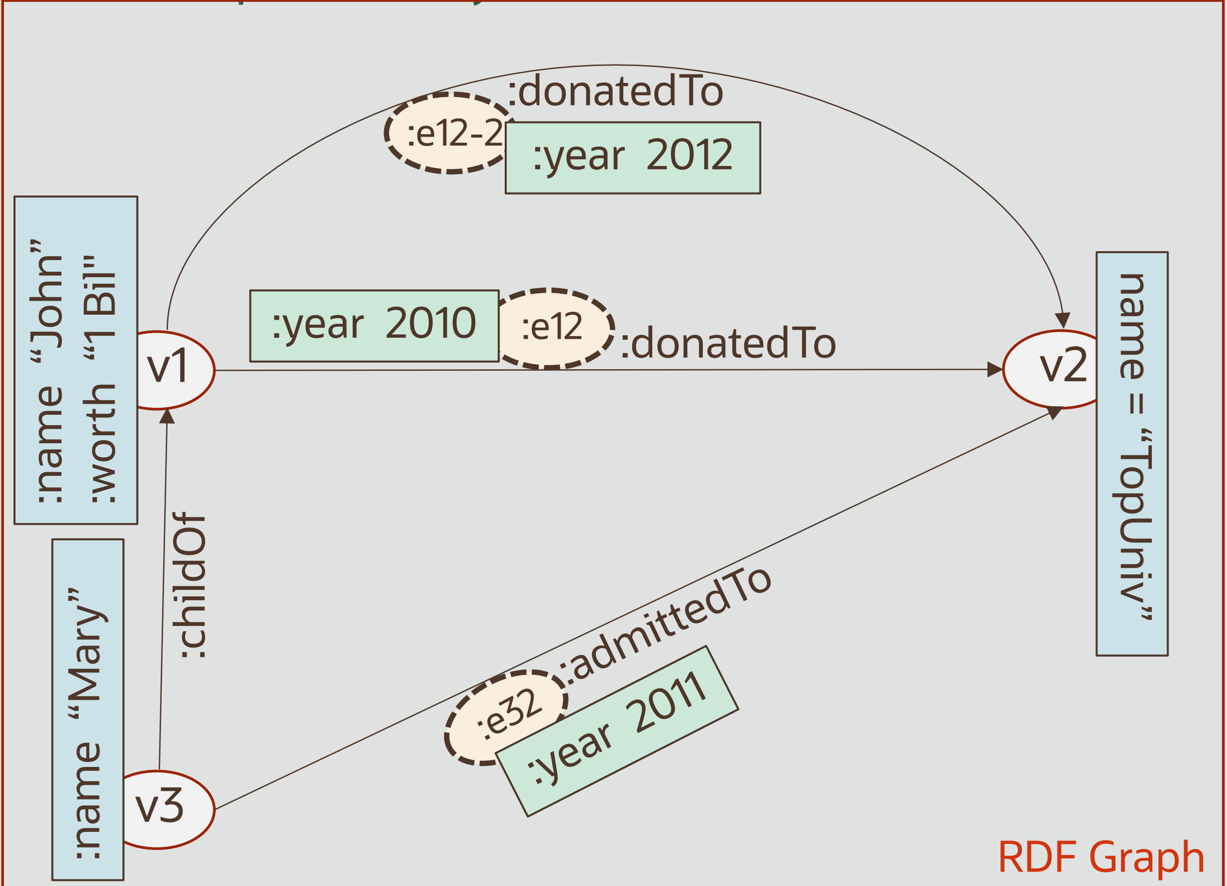
John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively.

Mary, a child of John, got admitted to Top University in 2011.

```
PREFIX orardf: <http://xmlns.oracle.com/orardf/>
select ?prop ?domain ?range
{graph orardf:schgraph> {
   ?prop       rdf:type rdf:Property ;
               orardf:includesDomainRange ?domrng .
   ?domrng  orardf:includesDomain         ?domain ;
               orardf:includesRange          ?range .
}}
```

RDB2RDF_METADATA=T

?prop	?domain	?range
:admittedTo	:Admitted	:University
:childOf	:Child	:Person
:donatedTo	:Donor	:University
:name	:Person, :University	xsd:string
:worth	:Person	xsd:string
:year	:Admission, :Donation	xsd:decimal

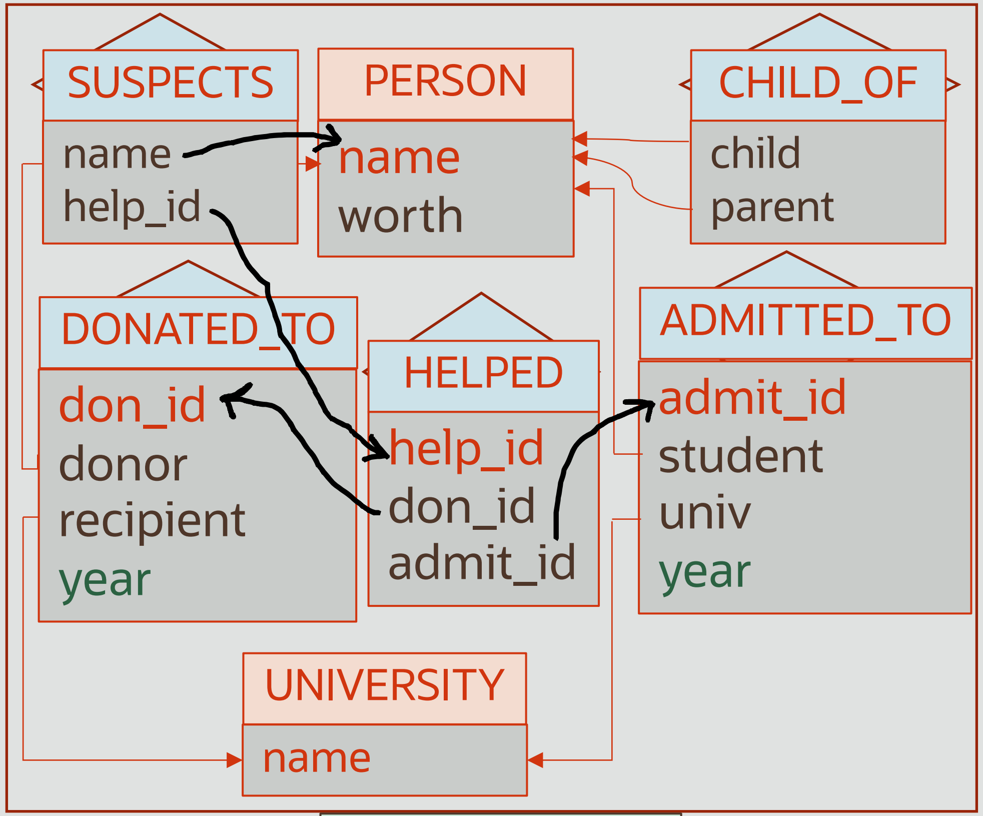


RDF Graph

Relational to RDF Quads:
Additional Data

…

Bob suspects that the 2010 donation helped the 2011 admission.



SUSPECTS
name
help_id

PERSON
name
worth

CHILD_OF
child
parent

DONATED_TO
don_id
donor
recipient
year

HELPED
help_id
don_id
admit_id

ADMITTED_TO
admit_id
student
univ
year

UNIVERSITY
name

ER diagram

SUSPECTS

name	help_id
Bob	1

HELPED

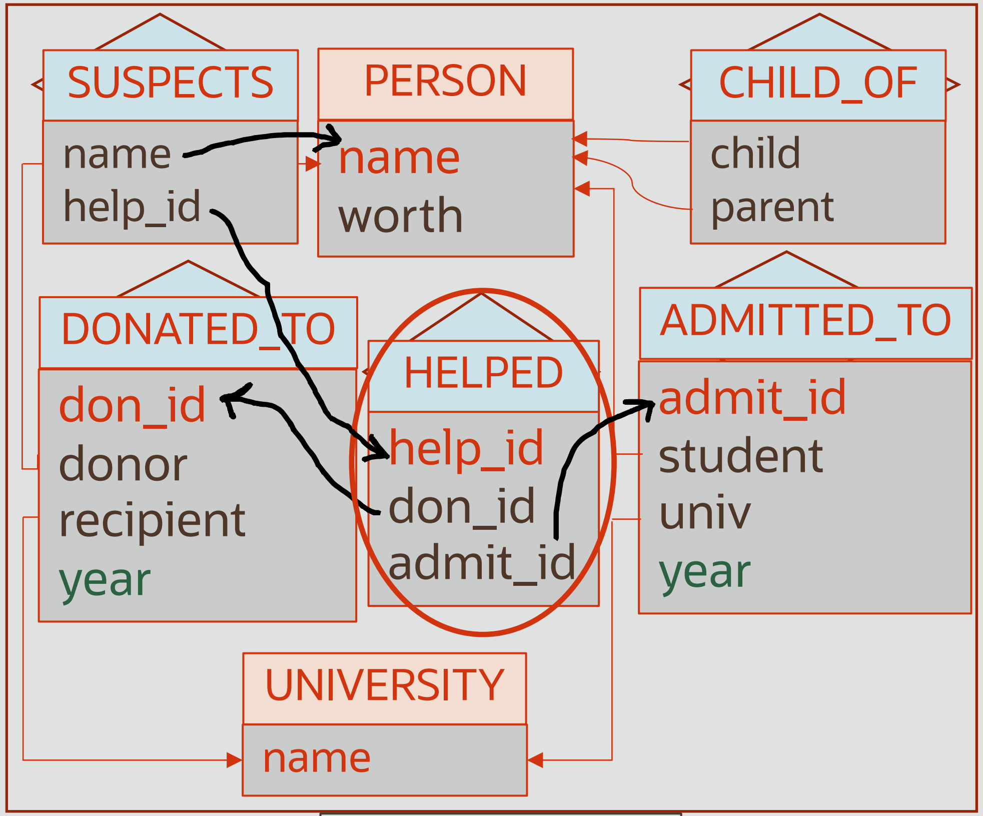
help_id	don_id	admit_id
1	1	1

Additional Relational Data

Relational to RDF Quads:
Relation-As-Entity in ER model ➔ R2RML

…

Bob suspects that the 2010 donation helped the 2011 admission.



ER diagram

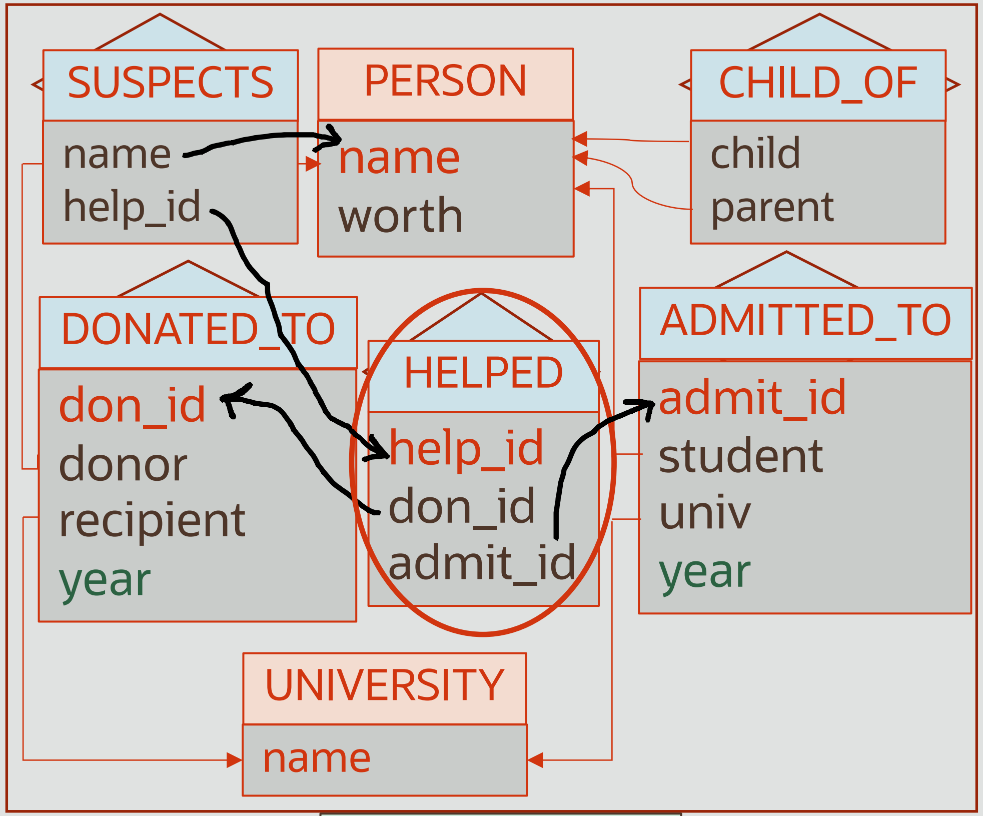
```
# -- HELPED (relationship-as-entity) table --
#
ex:TMap_HELPED_AS_ENTITY
  rr:logicalTable [ rr:tableName "RDFU.HELPED" ] ;

  rr:subjectMap [
     rr:template "http://ex/helpId#{HELP_ID}" ;
     rr:class ex:Helping ] ;
  …
```

R2RML mapping

Relational to RDF Quads:
Relation in ER model ➔ R2RML

…

… Bob suspects that the 2010 donation helped the 2011 admission.



ER diagram

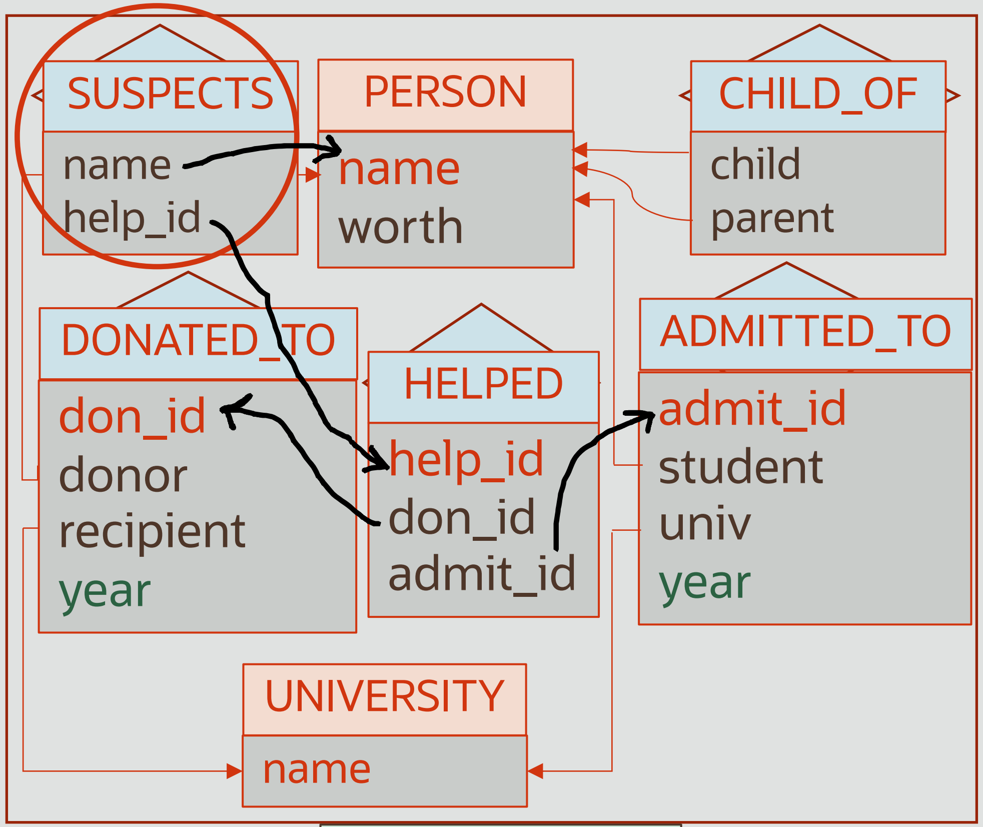
```
# -- HELPED (relationship) table --
# EDGE➔ (don_id)-[helped]->(admit_id)
#
ex:TMap_HELPED
  rr:logicalTable [ rr:tableName "RDFU.HELPED" ] ;

  rr:subjectMap [ rr:template "http://ex/donationId#{DON_ID}" ] ;

  rr:predicateObjectMap [
     rr:graphMap [ rr:template "http://ex/helpId#{HELP_ID}" ] ;
     rr:predicate :helped ;
     rr:objectMap [
       rr:parentTriplesMap ex:TMap_ADMITTED_TO_AS_ENTITY ;
       rr:joinCondition [ rr:child ”ADMIT_ID" ; rr:parent ”ADMIT_ID" ] ] ] .
```

R2RML mapping

Relational to RDF Quads:
Relation in ER model ➔ R2RML

…

… Bob suspects that the 2010 donation helped the 2011 admission.



ER diagram

```
# -- SUSPECTS table --
# EDGE➔ (name)-[suspects]->(help_id)
#
ex:TMap_SUSPECTS
  rr:logicalTable [ rr:tableName "RDFU.SUSPECTS" ] ;

  rr:subjectMap [ rr:template "http://ex/PERSON/{NAME}" ] ;

  rr:predicateObjectMap
    [ rr:predicate :name  ; rr:objectMap [ rr:column ”NAME"  ] ] ;
    [ rr:predicate :suspects ;
      rr:objectMap [
        rr:parentTriplesMap ex:TMap_HELPED_AS_ENTITY ;
        rr:joinCondition [ rr:child ”HELP_ID" ; rr:parent ”HELP_ID" ] ] ] .
```

R2RML mapping

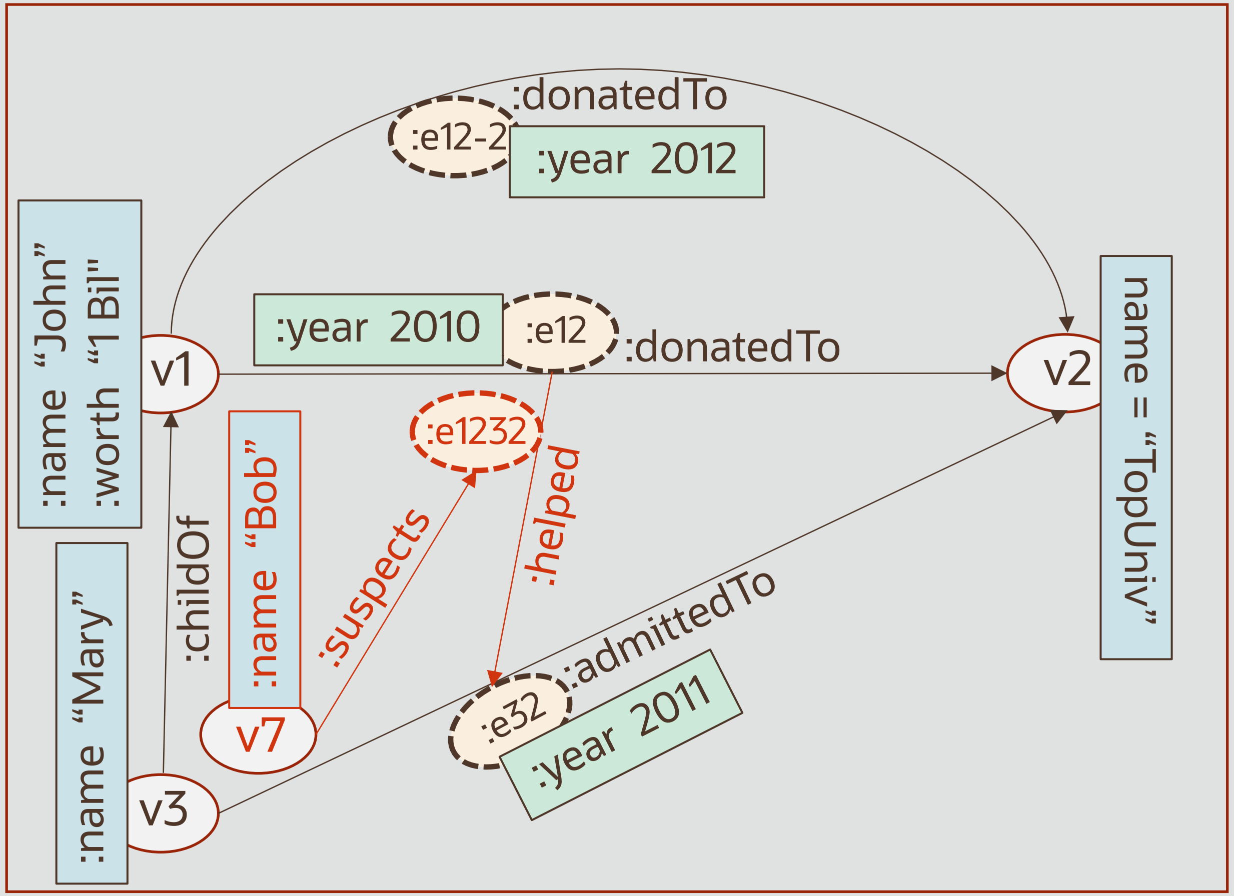
Relational to RDF Quads:
Schema for Resulting RDF Graph

John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively. Mary, a child of John, got admitted to Top University in 2011. Bob suspects that the 2010 donation helped the 2011 admission.

How was the schema affected? Two new properties got added!

?prop	?domain	?range
:admittedTo	:Admitted	:University
:childOf	:Child	:Person
:donatedTo	:Donor	:University
:helped	**:Donation**	**:Admission**
:suspects	**:Person**	**:Helping**
:name	:Person, :University	xsd:string
:worth	:Person	xsd:string
:year	:Admission, :Donation	xsd:decimal

Schema for RDF Data



:e12 ➔<http://ex/donationId#1>, :e12-2 ➔<http://ex/donationId#2>,
:e32 ➔<http://ex/admissionId#1>, :e1232 ➔<http://ex/helpId#1>.

Relational to RDF Quads:
Resulting RDF Graph, SPARQL query

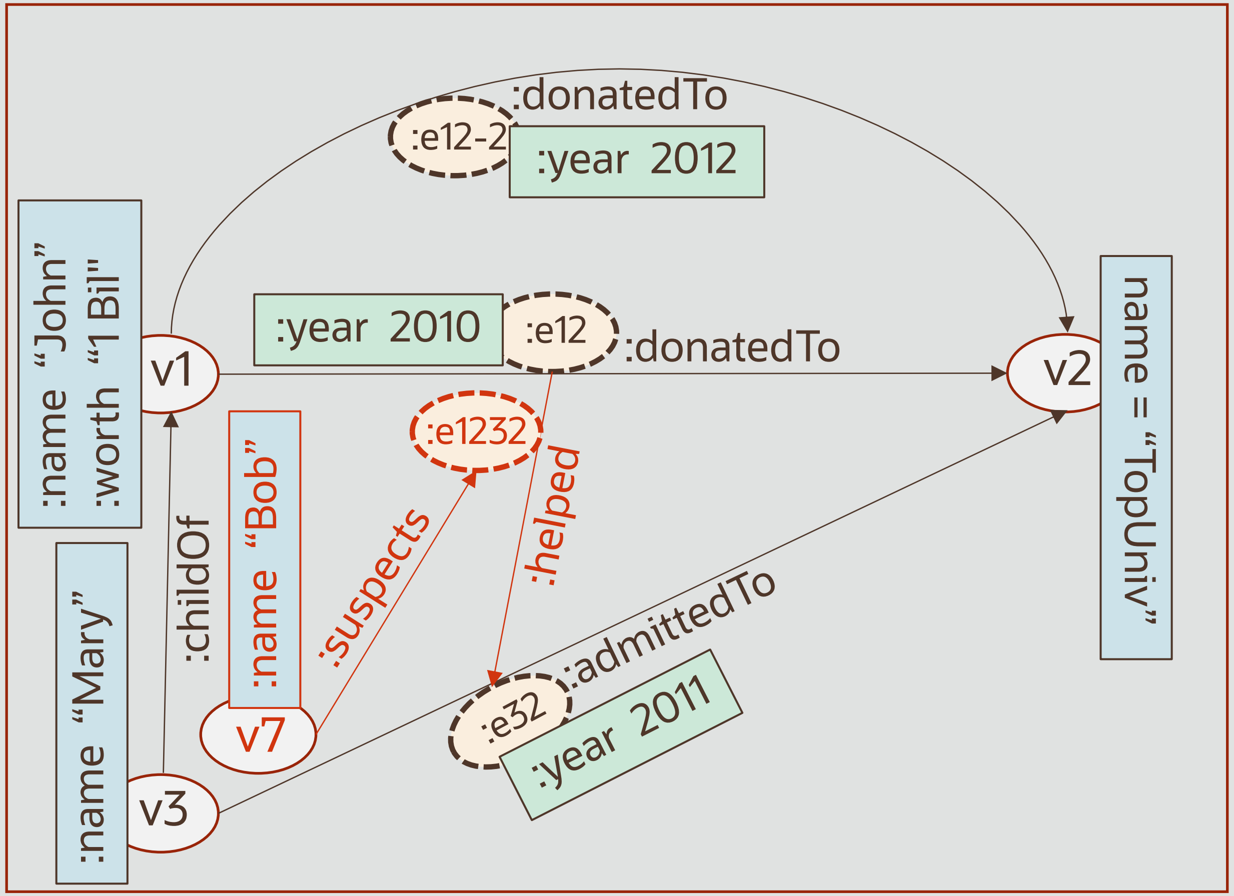
John, whose net worth is $1 billion, donated twice to Top University, in the years 2010 and 2012, respectively. Mary, a child of John, got admitted to Top University in 2011. Bob suspects that the 2010 donation helped the 2011 admission.

Find names of parent, university, and child where parent donated to the university during a year and his/her child got admitted to the university in the following year.

```
SELECT ?paName ?univName ?chName
WHERE {
  ?child     :childOf    ?parent .
  #
  graph ?donEdge { ?parent    :donatedTo    ?univ }
  ?donEdge    :year        ?donYear .
  #
  graph ?admEdge { ?child      :admittedTo   ?univ }
  ?admEdge   :year        ?admYear .
  #
  FILTER ( ?admYear = ?donYear + 1 )
  ?child     :name       ?chName .
  ?parent  :name       ?paName .
  ?univ      :name       ?univName }
```

All pre-existing queries remain valid.

SPARQL Query



:e12 ➔<http://ex/donationId#1>, :e12-2 ➔<http://ex/donationId#2>,
:e32 ➔<http://ex/admissionId#1>, :e1232 ➔<http://ex/helpId#1>.

Agenda

Part 1

- Backward Compatibility in Evolving Graphs
- Distinguishing among Graph Types
- Brief Intro to RDF
- Backward Compatibility: An Example and Demo

Part 2

- Intro to SPARQL Query and SPARQL Update
- Evolving Data: Movie Review Demo
- PGQL vs SPARQL
- Graph Analytics on RDF data
- Demo

Part 3

- Intro to R2RML
- Advanced Modeling using R2RML: An Example and Demo
- Baseball Data: A Real-World Example and Demo

Resources for Getting Started

- VM image: https://www.oracle.com/database/technologies/databaseappdev-vm.html
- Oracle Database Docker

  Single instance database from

  https://github.com/oracle/docker-images/tree/master/OracleDatabase
- Oracle Cloud

  Use **Oracle Database Cloud Service** with $300 free credits

  On the roadmap: RDF Graph support in 'Always Free Tier'

RDF View Demo Setup

Data: Baseball data source: http://baseball1.com/statistics (CSV files)

Entity tables:

```
create table people (playerID varchar2(30) primary key, birthYear varchar2(4),
            debut date, nameGiven varchar2(50), finalGame date) compress;

create table teams (yearID varchar2(4), teamID  varchar2(10), name varchar2(100),
            primary key (yearID, teamID)) compress;

create table schools (schoolID varchar2(30) primary key, name_full varchar2(100),
           city varchar2(30), tate varchar2(30), country varchar2(30)) compress;
```

RDF View Demo Setup

Relationship tables:

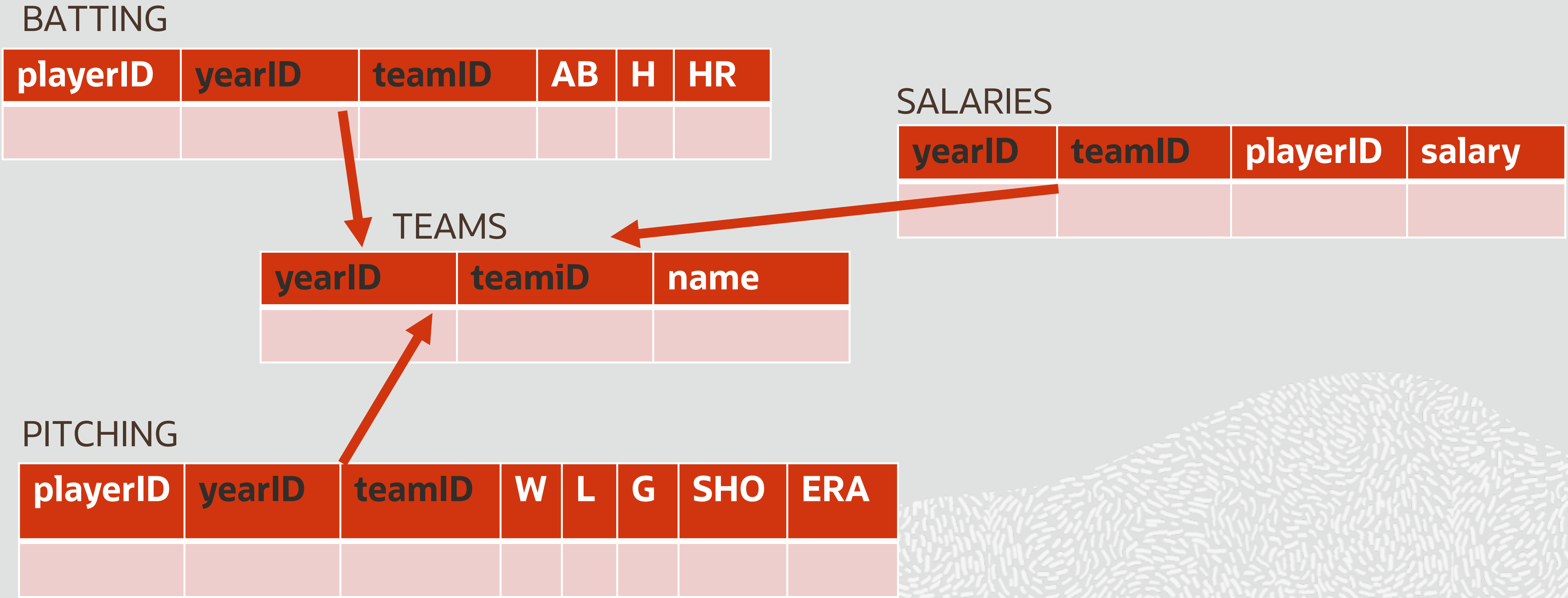
```
create table salaries (yearID varchar2(4), teamID varchar2(10), playerID varchar2(30),
                 salary int, primary key(yearID, teamID, playerID),
                foreign key(yearID, teamID) references TEAMS(yearID, teamID),
                foreign key (playerID) references PEOPLE(playerID)) compress;

create table batting (playerID varchar2(30) primary key, yearID varchar2(4),
               teamID varchar2(10), AB int, H int, HR int,
              foreign key(yearID, teamID) references TEAMS(yearID, teamID),
              foreign key(playerID) references PEOPLE(playerID)) compress;
```

RDF View Demo Setup

Relationship tables:

```
create table pitching (playerID varchar2(30) primary key, yearID varchar2(4),
        teamID varchar2(10), W int, L int, G int, SHO int, ERA number,
        foreign key(yearID, teamID) references TEAMS(yearID, teamID),
        foreign key(playerID) references PEOPLE(playerID));

create table CollegePlaying (playerID varchar2(30), schoolID varchar2(30), year varchar2(4),
        foreign key(playerID) references PEOPLE(playerID),
        foreign key(schoolID) references SCHOOLS(schoolID)) compress;
```

Relational Tables



BATTING

playerID	yearID	teamID	AB	H	HR

SALARIES

yearID	teamID	playerID	salary

TEAMS

yearID	teamiD	name

PITCHING

playerID	yearID	teamID	W	L	G	SHO	ERA

Data Loading

Oracle SQL*Loader: (people.ctl)

```
        load data into table people
        insert
        fields terminated by ”,"
        (
        playerID,
        birthYear,
        field1 FILLER,
        field2 FILLER,
        .........
        debut
         finalGame,
        f18 FILLER,
        ..
        )
```

Data Loading

Oracle SQL*Loader: (people.par)

```
userid=rdfuser@orcl/rdfuser
control=/home/oracle/people.ctl
log=demo.log
bad=demo.bad
data=/home/oracle/People.csv
direct=true
errors=10
```

Load command: sqlldr people.par

R2RML

```
DECLARE
  r2rmlStr CLOB;
BEGIN
  r2rmlStr :=
   '@prefix rr: <http://www.w3.org/ns/r2rml#>. '||
   '@prefix xsd: <http://www.w3.org/2001/XMLSchema#>. '||
   '@prefix : <http://demo/>. '||
   '@prefix ex: <http://ex/>. '||'
# -- PEOPLE table --
ex:TMap_PLAYERS
  rr:logicalTable [ rr:tableName "rdfuser.PEOPLE" ] ;
  rr:subjectMap [ rr:template "http://ex/PLAYER/{playerID}" ; rr:class :Player ] ;
# -- generate triples for scalar columns
  rr:predicateObjectMap
    [ rr:predicate :givenName ; rr:objectMap [ rr:column "nameGiven"  ] ]
  , [ rr:predicate :birthYear ; rr:objectMap [ rr:column "birthYear" ] ] .
```

R2RML

```
#
# -- TEAMS table --
#
ex:TMap_TEAMS
  rr:logicalTable [ rr:tableName "rdfuser.TEAMS" ] ;
  rr:subjectMap [ rr:template "http://ex/TEAM/{yearID}-{teamID}" ; rr:class :Team ] ;
# -- generate triples for scalar columns
  rr:predicateObjectMap [ rr:predicate :name  ; rr:objectMap [ rr:column "NAME"  ] ] .
```

R2RML

```
#
# -- BATTING table --
#
ex:TMap_TEAM_PLAYER_BATTING
  rr:logicalTable [ rr:tableName "rdfuser.BATTING" ] ;
  rr:subjectMap [ rr:template "http://ex/PLAYER/{yearID}-{teamID}-{playerID}" ; rr:class
:BattingInfo ] ;
# -- generate triples for scalar columns
  rr:predicateObjectMap
    [ rr:predicate :atBat ; rr:objectMap [ rr:column "AB" ] ]
  , [ rr:predicate :hits ; rr:objectMap [ rr:column "H" ] ]
  , [ rr:predicate :homeRuns ; rr:objectMap [ rr:column "HR" ] ] .
```

R2RML

```
#
# -- PITCHING table --
#
ex:TMap_TEAM_PLAYER_PITCHING
  rr:logicalTable [ rr:tableName "rdfuser.PITCHING" ] ;
  rr:subjectMap [ rr:template "http://ex/PLAYER/{yearID}-{teamID}-{playerID}" ; rr:class
:PitchingInfo ] ;
# -- generate triples for scalar columns
  rr:predicateObjectMap
    [ rr:predicate :wins ; rr:objectMap [ rr:column "W" ] ]
  , [ rr:predicate :losses ; rr:objectMap [ rr:column "L" ] ]
  , [ rr:predicate :games ; rr:objectMap [ rr:column "G" ] ]
  , [ rr:predicate :shutOuts ; rr:objectMap [ rr:column "SHO" ] ]
  , [ rr:predicate :earnedRunAvg ; rr:objectMap [ rr:column "ERA" ] ] .
```

R2RML

```
#
# -- SALARIES (relationship) table --
#
ex:TMap_PLAYED_SALARY
  rr:logicalTable [ rr:tableName "rdfuser.SALARIES" ] ;
  rr:subjectMap [ rr:template "http://ex/PLAYER/{playerID}" ; rr:class :SalariedPlayer ] ;
# -- generate the relationship triples
  rr:predicateObjectMap [ rr:graphMap [ rr:template "http://ex/PLAYER/{yearID}-{teamID}-{playerID}" ]
;
                          rr:predicate :playedFor ;
                          rr:objectMap [ rr:parentTriplesMap ex:TMap_TEAMS ;
                                         rr:joinCondition
                                           [ rr:child "yearID" ; rr:parent "yearID" ]
                                         , [ rr:child "teamID" ; rr:parent "teamID" ] ] ] .
```

R2RML

```
# -- BATTING (relationship) table --
#
ex:TMap_PLAYED_BATTER
 rr:logicalTable [ rr:tableName "rdfuser.BATTING" ] ;
 rr:subjectMap [ rr:template "http://ex/PLAYER/{playerID}" ; rr:class :Batter ] ;
# -- generate the relationship triples
 rr:predicateObjectMap [ rr:graphMap [ rr:template "http://ex/PLAYER/{yearID}-{teamID}-{playerID}" ]
;
             rr:predicate :playedFor ;
             rr:objectMap [ rr:parentTriplesMap ex:TMap_TEAMS ;
                     rr:joinCondition
                      [ rr:child "yearID" ; rr:parent "yearID" ]
                     , [ rr:child "teamID" ; rr:parent "teamID" ] ] ] .
```

R2RML

```
#
# -- PITCHING (relationship) table --
#
ex:TMap_PLAYED_PITCHER
  rr:logicalTable [ rr:tableName "rdfuser.PITCHING" ] ;
  rr:subjectMap [ rr:template "http://ex/PLAYER/{playerID}" ; rr:class :Pitcher ] ;
# -- generate the relationship triples
  rr:predicateObjectMap [ rr:graphMap [ rr:template "http://ex/PLAYER/{yearID}-{teamID}-
{playerID}" ] ;
                          rr:predicate :playedFor ;
                          rr:objectMap [ rr:parentTriplesMap ex:TMap_TEAMS ;
                                         rr:joinCondition
                                           [ rr:child "yearID" ; rr:parent "yearID" ]
                                         , [ rr:child "teamID" ; rr:parent "teamID" ] ] ] .
```

R2RML

```
#
# -- SALARIES table --
#
ex:TMap_TEAM_PLAYER_SALARY
  rr:logicalTable [ rr:tableName "rdfuser.SALARIES" ] ;
  rr:subjectMap [ rr:template "http://ex/PLAYER/{yearID}-{teamID}-{playerID}" ; rr:class
:SalaryInfo ] ;
# -- generate the relationship triples
  rr:predicateObjectMap
    [ rr:predicate :salary ; rr:objectMap [ rr:column "salary" ] ] .
```

Create RDFView

SGA setting:

```
        conn sys/oracle@orcl as sysdba
        create pfile='/home/oracle/rdf_init.ora' from spfile;
```

Edit rdf_init.ora and set: sga_target=2G

Restart DB:

```
        conn sys/oracle@orcl as sysdba
        alter session set container=CDB$ROOT;
        shutdown immediate
        conn sys/oracle@//localhost:1521/orclcdb as sysdba;
        startup pfile= /home/oracle/rdf_init.ora
```

```
sem_apis.create_rdfview_model(
  model_name => ‘rdfview_demo_graph’,  tables => NULL,
  r2rml_string => r2rmlStr,            r2rml_string_fmt => 'TURTLE’,
  network_owner=>'RDFUSER’,     network_name=>'NET1'
 );
```

ORACLE