

CSCE60413 - Semantic Web

Web versus Semantic Web

Adila Krisnadhi





UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Hello!



I am Adila Krisnadhi

Faculty member at the Faculty of Computer Science, Universitas
Indonesia.

Co-director of Tokopedia-UI AI Center of Excellence
Ontology engineer and Semantic Web enthusiast



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Credits

- ❖ Presentation template by [SlidesCarnival](#)
- ❖ Photographs by [Unsplash](#)
- ❖ Backgrounds by [SubtlePatterns](#)
- ❖ Content by Eero Hyvonen from the CS-E410 lecture slides “WWW Today” and “Semantic Web” (with partial adaptation by me).



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF

COMPUTER
SCIENCE

1.

The World Wide Web



UNIVERSITAS
INDONESIA

Virtute, Prodigio, Justitia

FACULTY OF
COMPUTER
SCIENCE

The Web in a nutshell



Users

- 7.676 billion world population.
- 4.4 billion Internet users.
- 5.1 billion unique mobile users.



Content

- Approx. 55 billion pages indexed by Google (size of Surface Web).
- Deep Web (not just Dark Web) is est. up to 400 times larger



Effectiveness of publishing

- All information readable by everyone.
- New content easy to publish to billions of audiences.
- Almost “free” usage.



UNIVERSITAS
INDONESIA
Virtute, Prodigio, Justitia

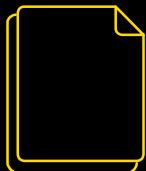
FACULTY OF
COMPUTER
SCIENCE

Basic ingredients of the Web

<<http://www.blablabla.com/myfolder/mydoc.html>>

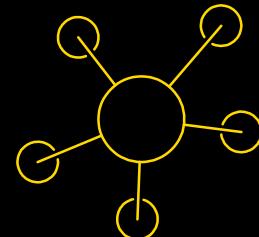
URL addresses

Represent resources such as websites, documents, pictures



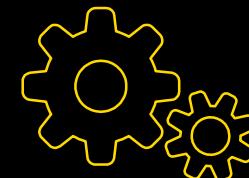
HTML language

Expresses resource content (Web pages)
Enables hyperlinks



HTTP protocol

Specifies mechanism for transferring resources between server and client.





UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF

COMPUTER
SCIENCE

2. Services on the web



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

The Web provides the following services ...

Functional
Banking, shopping,
government,
entertainment, etc.

Information retrieval
Search engines and
browsing.
Portals, directories.
Databases in different
applications.



UNIVERSITAS
INDONESIA

Veritas. Virtus. Veritate.

FACULTY OF
COMPUTER
SCIENCE

Information retrieval challenges: end-user perspective

- ❖ **Search query formulation**
 - ❖ Creating queries that work as intended is not always easy.
- ❖ **Search result quality**
 - ❖ Recall: proportion of the relevant information is found.
 - ❖ Precision: proportion of the found information is relevant.
 - ❖ Relevance: how well the results correspond to the user needs
- ❖ **Search result presentation**
 - ❖ Ease of understanding
 - ❖ Ranking and structuring



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Limitation of basic text search: Examples (1)

Search term may appear in an irrelevant document

- ❖ “Halaman ini *tidak berbicara tentang kucing.*”

Identifying (some) synonyms

- ❖ “hewan” vs. “satwa”, “pranala” vs. “tautan”
 - Bad recall (relevant pages not found), query formulation difficult

Identifying homonyms

- ❖ “Palu”: a city, a tool
- ❖ “Depok”: a city south of Jakarta, a district in Yogyakarta
 - Bad precision, garbage results; results hard to understand; query formulation hard



UNIVERSITAS
INDONESIA

Veritas. Pudicit. Intellec.

FACULTY OF
COMPUTER
SCIENCE

Limitation of basic text search: Examples (2)

Computer does not understand relations between concepts

- ❖ Narrower-broader concept, part-whole (Google is smarter now)
 - Search “surface depression in solar system” should return all craters on Earth.
- ❖ Missing background or common sense knowledge
 - Searching ‘smoke’ may not return pages about ‘fire’

Information to be searched is fragmented, but results cannot be aggregated

- ❖ “Search publications of the members of the research group X”



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Limitation of basic text search: Examples (3)

Finding relations between information resources is hard

- ❖ "How is Dieng related to Borobudur?"

Search does not go beyond “memorizing”

- ❖ How much does a kilogram of feathers weigh on the moon?
- ❖ With lots of memorizing, problem solving resembles remembering.

Context/personalization is under-utilized

- ❖ What could I do today in Depok?

Computer unable to “understand” meaning/semantics

- ❖ Which city has Makassar as one of its district?



UNIVERSITAS
INDONESIA

Veritas. Pudicit. Justitia.

FACULTY OF
COMPUTER
SCIENCE

Browsing challenges in the Web: end-user perspective

- ❖ **Understanding the “big picture” in a large fragmented information space**
 - ❖ “Lost in the hyperspace”
- ❖ **Links get out of date and destroyed**
 - ❖ The linked target pages expire or are removed entirely
 - ❖ New pages do not get linked to old ones
 - ❖ Old pages do not get linked to new ones
- ❖ **Reliability of information and their providers**
 - ❖ “Web of trust”
 - ❖ “Flat Earth” organization’s page vs. our university’s scientific page
 - ❖ Wikipedia vs. Encyclopedia Britannica



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Knowledge management challenges: Information provider perspective

- ❖ **Structuring contents with links is manual work**
 - ❖ Information does not get linked at content level without human effort
- ❖ **Different organizations create overlapping information**
 - ❖ The same work is done multiple times
- ❖ **The contents and their structures are not interoperable**
 - ❖ Aggregation of collections of different memory organizations is difficult
 - ❖ Lack of interoperability prevents combining of contents
 - ❖ Lack of interoperability prevents the management of contents
- ❖ **Information about the contents and their changes is not communicated between organizations**



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

3. Machine Processability (and Understandability?)



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Web content is mainly created for ...

- ❖ Humans

- ❖ Machines



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

What can machines do w.r.t. information on the Web?

- ❖ Mediate
- ❖ Display
- ❖ Understand



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Machine processability problem

- ❖ Web content → for humans
- ❖ Machines only mediate and display content, and do not understand Web content.
- ❖ A web service → machine to help human
 - But requires machine to understand Web content.



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

How can we build a more intelligent Web?

- ❖ Make applications to be more intelligent?
- ❖ Represent content in a more intelligent way?



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Smart application approach

Machines need to be able to understand/interpret content, but:

- ❖ Automated interpretation of natural language is difficult
- ❖ Non-textual content is very hard to interpret
- ❖ Interpretation often needs context and common sense.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Smart content approach

- ❖ Information is provided in a way that allows machines to understand it.
 - → **Key idea of Semantic Web**
- ❖ Who provides the information in that form?
 - → Human (and machines)
- ❖ History:
 - W3C Semantic Web activity in 2001
 - W3C Web Services activity in 2002



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE



“Semantic” in Semantic Web
means “understandable” to
machines



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF

COMPUTER
SCIENCE

4. Data (Knowledge?) Representation



UNIVERSITAS
INDONESIA
Virtute, Prodigio, Inclita

FACULTY OF
COMPUTER
SCIENCE

Markup Languages: Main Ideas

Domain- and environment-independent standard for documents

Creation

Documents are text files

Management

Transferring

Open, simple format

Usable on all platforms

Easy modification, storing, reading, transfer

Future-proof

Separate structure, content, and presentation

Describing document structure (for programmer) → HTML: <h1> Heading </h1>

Describing information content (for programmer) → XML: <address>Jl. Margonda</address>

Presentation is decided by the reader (browser)



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

The case for XML

- ❖ **Utilization of content structure**
 - ❖ Better recall/precision by search engines
- ❖ **Syntax validation**
- ❖ **Widely used on the Web**
 - ❖ Encoding of knowledge in open format
 - ❖ Open APIs for programming languages allow programmatic processing of the pages' content
- ❖ **Vendor-independent**
- ❖ **File format is unlikely to change**
 - ❖ Pages are text files
- ❖ **Enables domain-specific standard languages to be built on top**



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Syntactic interoperability through standardization

- ❖ World Wide Web Consortium (W3C)
 - ❖ Cooperation body of vendors, operators, universities, etc.
 - ❖ Creates and maintains Web standards/recommendations (HTML, XML, etc.)
- ❖ Domain-specific organizations
 - ❖ ISO: various domains, except electrical/electronical
 - ❖ IEC: electronical
 - ❖ CEN: european body, various domains
 - ❖ UN/CEFACT: electronic
 - ❖ OASIS: IT-related
 - ❖



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Case against XML

- ❖ Difficult to read and process by humans
 - ❖ Notation is not human-friendly
- ❖ Documents contain lots of repetition
 - ❖ Lots of redundancy, e.g., start and end tags, blowing up the size of the markup files.
 - ❖ Laborious to write.
 - ❖ Needs more bandwidth for transfer



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Alternative (better) notations

- ❖ JSON (JavaScript Object Notation)
 - ❖ Representation as nested key-value pairs
 - ❖ Integrated into JavaScript: easy, efficient to use
 - ❖ Widely adopted
 - ❖ Standardized by IETF (RFC) and ISO/IEC
 - ❖ Adopted and extended into Semantic Web as JSON-LD

- ❖ Simple Semantic Web notations
 - ❖ Turtle and various OWL notations
 - ❖ Widely used
 - ❖ Standardized by W3C



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

5. Semantic Web: Why Syntax only is not Enough



UNIVERSITAS
INDONESIA

Veritas. Pudicit. Justitia.

FACULTY OF
COMPUTER
SCIENCE

Why syntax-only content is limited



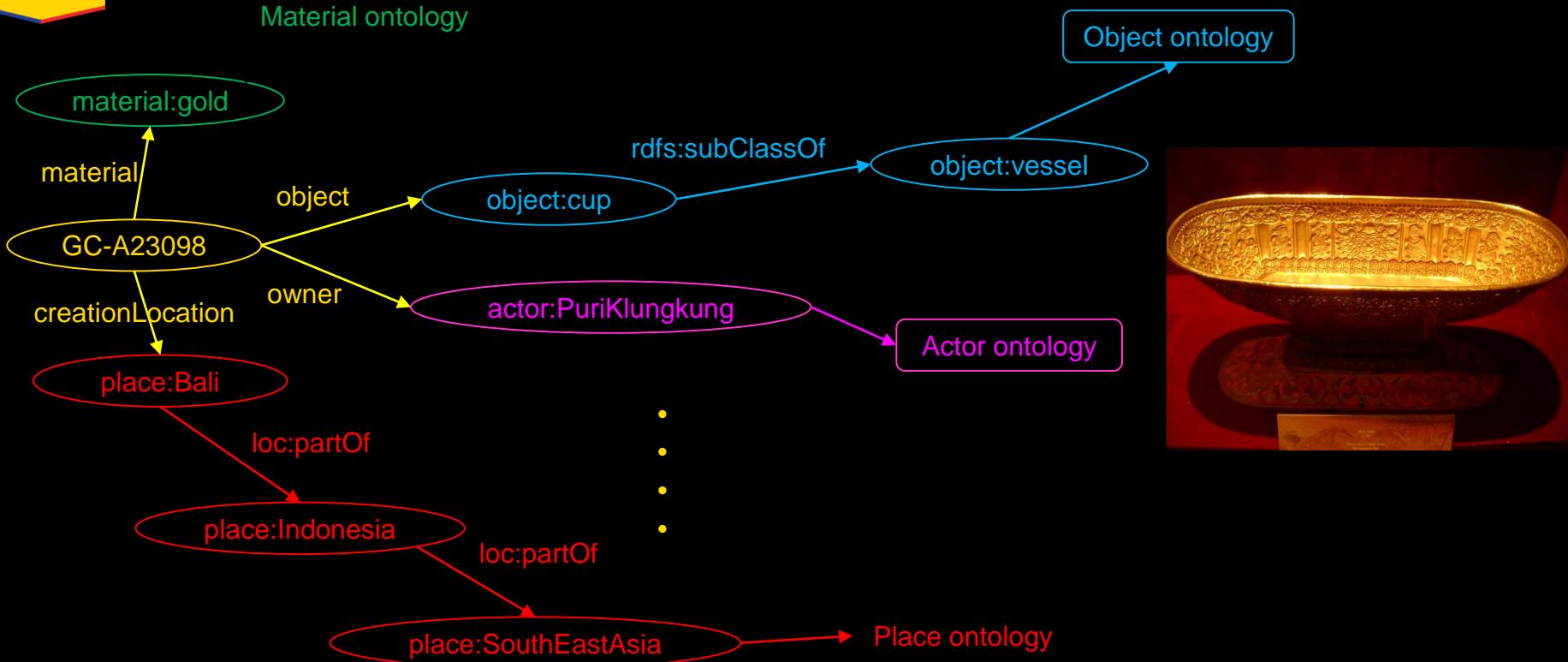
```
<artifact>
  <id>GC:A23098</id>
  <target>cup</target>
  <material>gold</material>
  <creationLocation>Bali</creationLocation>
  <originalOwner>Puri Klungkung</originalOwner>
</artifact>
```

Metadata cannot answer the following:

- Find all vessels?
- Find all metal product?
- Find all artifacts manufactured in South East Asia?
- Does Puri Klungkung own metal artifacts?



Key approach in Semantic Web: Use (Knowledge) Graph





UNIVERSITAS
INDONESIA

Veritas. Pudicit. Justitia.

FACULTY OF
COMPUTER
SCIENCE

Key approach in Semantic Web: Use (Knowledge) Graph



GC-A23098

```
rdfs:label "Cup" ;
:object object:cup ;
:material material:gold ;
:creationLocation place:Bali ;
:originalOwner actor:PuriKlungkung .
```

```
object:cup rdfs:subClassOf object:vessel .
```

```
place:Bali loc:partof place:Indonesia .
```

```
place:Indonesia loc:partOf place:SouthEastAsia .
```

-
-
-
-



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Why XML (and JSON) only is not enough

- ❖ Interpretation of XML languages need to be defined in domain-specific way.
- ❖ Integration of XML documents is cumbersome.
- ❖ Semantic of XML tags is only in human brain
 - Machines only see arbitrary symbols.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

So, we need ...

- ❖ A markup language whose interpretation is:
 - commonly agreed
 - shared across different application domains
 - machine-understandable

Semantic Web solution → Resource Description Framework (RDF)



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Semantic Web original vision

A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities ...

Tim Berners-Lee, James Hendler and Ora Lassila (Scientific American, May 2001)

<https://www.scientificamerican.com/article/the-semantic-web/>

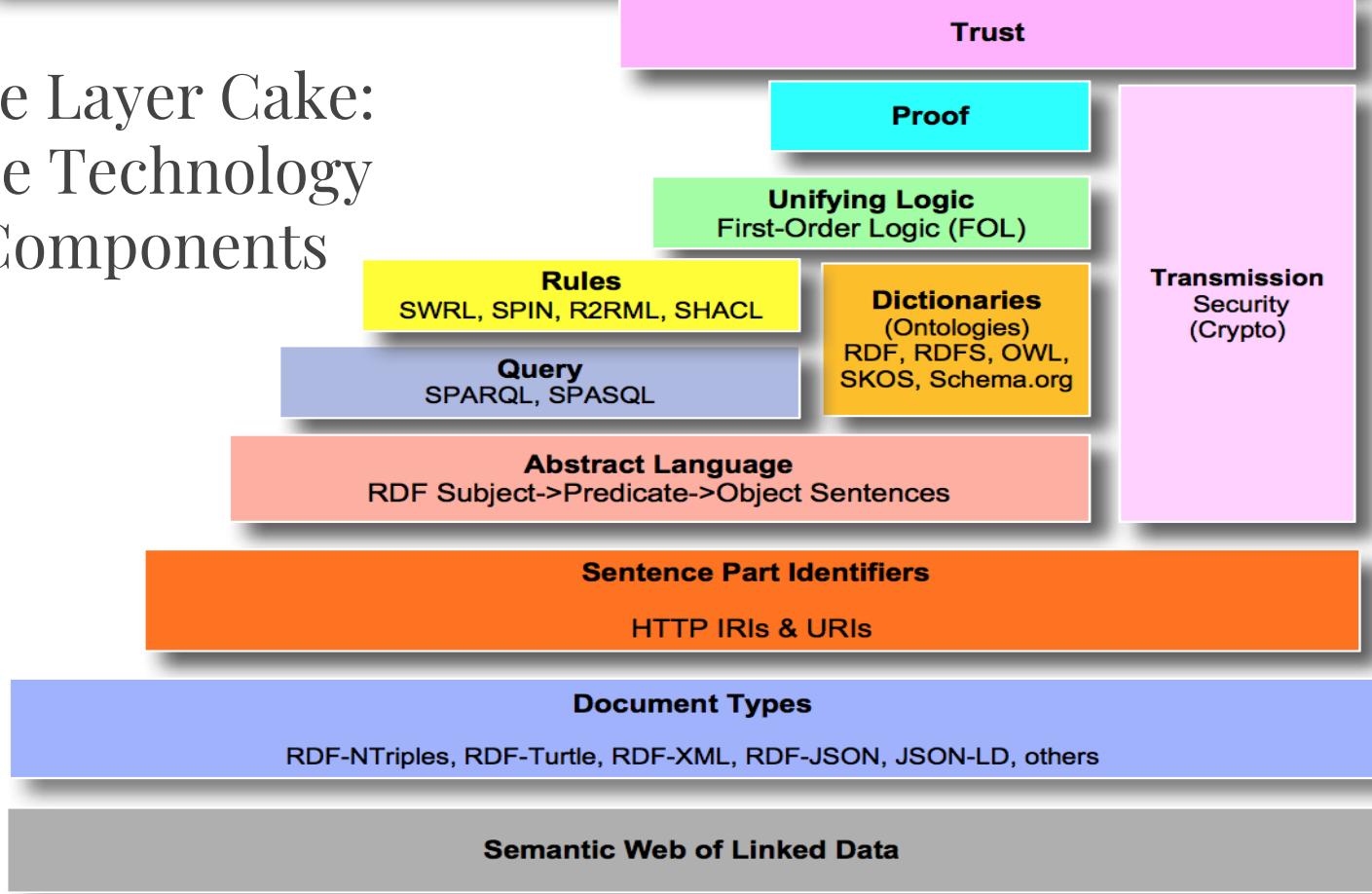


UNIVERSITAS
INDONESIA
Veritas Prendat Inedita

FACULTY OF
COMPUTER
SCIENCE

Smart (Cognitive) Applications & Services

The Layer Cake: The Technology Components





UNIVERSITAS
INDONESIA
Virtus, Prodigia, Inclusio

FACULTY OF
COMPUTER
SCIENCE

So, what is Semantic Web?

- ❖ **Content-perspective:** a new metadata layer on the web describing its contents in terms of shared vocabularies
 - Web as a global database system
 - Web of data (instead of Web of documents)
- ❖ **Application perspective:** machine-understandable web
 - Meaning/semantics of content accessible by machines.
 - Intelligent web services
 - Semantic interoperability
- ❖ **Technology perspective:** the layer(s) above XML.



UNIVERSITAS
INDONESIA

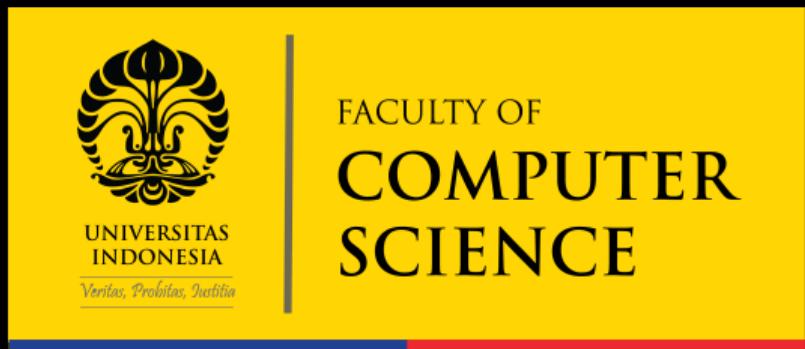
Veritas, Pudicit, Justitia

FACULTY OF

COMPUTER
SCIENCE



Fin.



Semantic Web o2: Graph-based Data Models

Adila Krisnadhi – adila@cs.ui.ac.id Faculty of Computer Science, Universitas
Indonesia

Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

Knowledge graph (KG): Definition

- The term “knowledge graph” has been used in the literature since 1973 with many, sometimes conflicting, definitions proposed. (See discussion on Appendix A of the KG Book).
- A **knowledge graph** is a **graph of data** intended to accumulate and convey **knowledge of the real world**, whose **nodes represent entities** of interest and whose **edges represent relations** between these entities.
 - The **data graph** conforms to a graph-based data model.
 - Knowledge refers to something that is **known**, perhaps accumulated from external sources, or extracted from the graph itself.

“Knowledge” in KG

- May be composed of
 - **simple** statements, e.g., “Jakarta is the capital of Indonesia”
 - simple statements can form edges in the data graph
 - **quantified** statements, e.g., “all capitals are cities”
 - require more expressive way to represent knowledge, e.g., using **ontologies** or **rules**.
- To entail and accumulate more knowledge, **deductive** and **inductive methods** can be used
 - e.g., we can deduce that the statement “Jakarta is a city” is also true.

Diversity and knowledge change in KG

- The structure and granularity of a KG can be highly diverse since it may be assembled from multiple sources.
- To handle the diversity, we use:
 - **schema**: high-level structure for the KG;
 - **identity**: denotes which nodes in the KG (or in external sources) refer to the same real-world entity;
 - **context**: indicates a specific setting in which some parts of the KG is held true.
- Knowledge changes over time, hence we study these effective methods for KG:
 - extraction
 - enrichment
 - quality assessment
 - refinement

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.

KG in practice

- **Open knowledge graphs:** all their content accessible online for the public good.
 - DBpedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)

KG in practice

- **Open knowledge graphs:** all their content accessible online for the public good.
 - DBpedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
 - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.

KG in practice

- **Open knowledge graphs:** all their content accessible online for the public good.
 - DBpedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
 - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.
- **Enterprise knowledge graphs:** internal for a particular organization/company and often applied for commercial use-cases, such as KGs created by:

KG in practice

- **Open knowledge graphs:** all their content accessible online for the public good.
 - DBpedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
 - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.
- **Enterprise knowledge graphs:** internal for a particular organization/company and often applied for commercial use-cases, such as KGs created by:
 - Google, Bing, Airbnb, Amazon, eBay, Uber, Facebook, LinkedIn, Accenture, Banca d'Italia, Bloomberg, Capital One, Wells Fargo, etc.

KG in practice

- **Open knowledge graphs:** all their content accessible online for the public good.
 - DBpedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
 - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.
- **Enterprise knowledge graphs:** internal for a particular organization/company and often applied for commercial use-cases, such as KGs created by:
 - Google, Bing, Airbnb, Amazon, eBay, Uber, Facebook, LinkedIn, Accenture, Banca d'Italia, Bloomberg, Capital One, Wells Fargo, etc.
- Applications of KG include search, recommendation systems, personal agents, advertising, business analytics, risk assessment, automation, etc.

Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

Motivating example

- A tourism board intends to model relevant data about attractions, services, and events. They start with a tabular structure, specifically with the following initial relational schema:

$Event(name, venue, type, \underline{start}, \underline{end})$

where *name* and *start* form the primary key to uniquely identify recurring events.

Motivating example

- A tourism board intends to model relevant data about attractions, services, and events. They start with a tabular structure, specifically with the following initial relational schema:

$Event(name, venue, type, \underline{start}, \underline{end})$

where *name* and *start* form the primary key to uniquely identify recurring events.

- What do you think the problems will be when the board starts to populate the table with data?

Motivating example (contd.)

Problems with

$Event(\underline{name}, \underline{venue}, \underline{type}, \underline{start}, \underline{end})$

Motivating example (contd.)

Problems with

$Event(\underline{name}, \underline{venue}, \underline{type}, \underline{start}, \underline{end})$

- Events may have multiple names.

Motivating example (contd.)

Problems with

$Event(\underline{name}, \underline{venue}, \underline{type}, \underline{start}, \underline{end})$

- Events may have multiple names.
- Events may have multiple venues.

Motivating example (contd.)

Problems with

$Event(\underline{name}, \underline{venue}, \underline{type}, \underline{start}, \underline{end})$

- Events may have **multiple names**.
- Events may have **multiple venues**.
- The board may **not yet know the start and end date-times** of future events.

Motivating example (contd.)

Problems with

$Event(\underline{name}, \underline{venue}, \underline{type}, \underline{start}, \underline{end})$

- Events may have multiple names.
- Events may have multiple venues.
- The board may not yet know the start and end date-times of future events.
- Events may have have multiple types.

Motivating example (contd.)

Problems with

$Event(\underline{name}, \underline{venue}, \underline{type}, \underline{start}, \underline{end})$

- Events may have multiple names.
- Events may have multiple venues.
- The board may not yet know the start and end date-times of future events.
- Events may have have multiple types.

What would the board do?

Motivating example (contd.)

The board would do the following:

- Create **internal identifiers** for events, say *id*.

Motivating example (contd.)

The board would do the following:

- Create **internal identifiers** for events, say *id*.
- **Adapt the relational schema** (e.g., perform normalization) to yield tables like the following:

$\text{EventName}(\underline{id}, \underline{\text{name}})$ $\text{EventStart}(\underline{id}, \text{start})$ $\text{EventEnd}(\underline{id}, \text{end})$
 $\text{EventVenue}(\underline{id}, \underline{\text{venue}})$ $\text{EventType}(\underline{id}, \underline{\text{type}})$

Note: Why do we make *name*, *venue*, and *type* primary keys?

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
 - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
 - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
 - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
 - because they do not know, in advance, what data to be modeled or what sources to be used.

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
 - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
 - because they do not know, in advance, what data to be modeled or what sources to be used.
- Once the latter/adapted relational schema is obtained, further integration of new data sources could be done without more changes.

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
 - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
 - because they do not know, in advance, what data to be modeled or what sources to be used.
- Once the latter/adapted relational schema is obtained, further integration of new data sources could be done without more changes.
 - The latter schema only makes minimal assumptions on multiplicities (1-1, 1-n, etc.)

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
 - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
 - because they do not know, in advance, what data to be modeled or what sources to be used.
- Once the latter/adapted relational schema is obtained, further integration of new data sources could be done without more changes.
 - The latter schema only makes minimal assumptions on multiplicities (1-1, 1-n, etc.)
 - The latter schema ends up with a set of binary relations between entities

Instances of binary relations naturally form a graph.

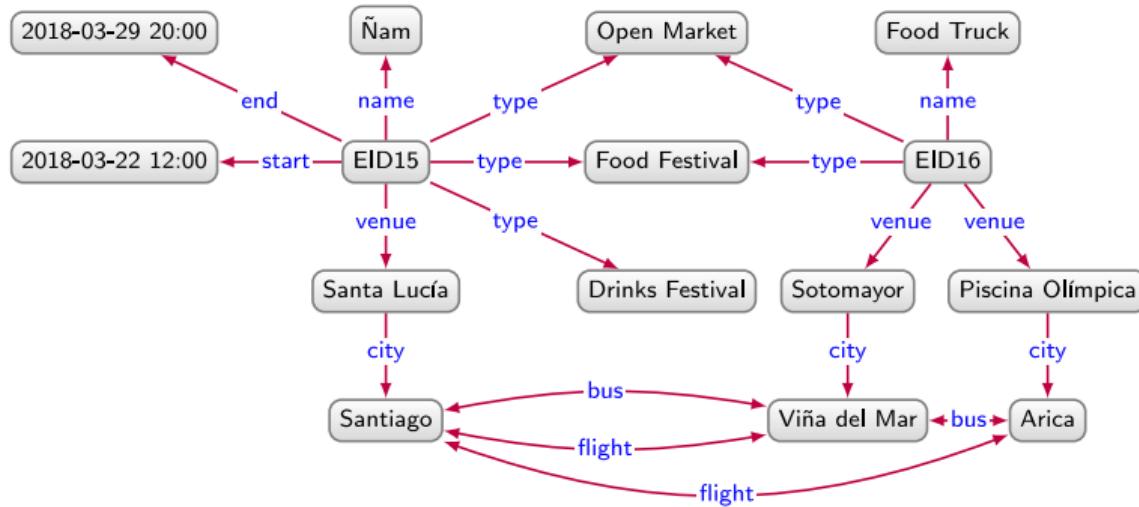
Instances of binary relations naturally form a graph.

So, why not use **graph** from the start?

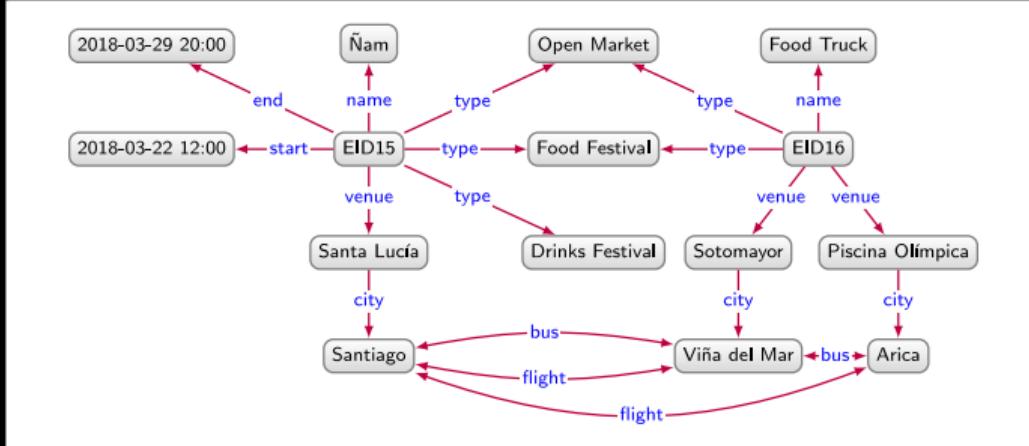
Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

DELG model about events



DELG model about events



- Does adding information to such a graph easy? How?
- Can we represent incomplete information in the graph? How?
- Do we need to define a schema upfront like relational databases?
- Do we require the data to be organized hierarchically like XML or JSON?

Directed edge-labeled graphs

Let Con be a countably infinite set of constants.

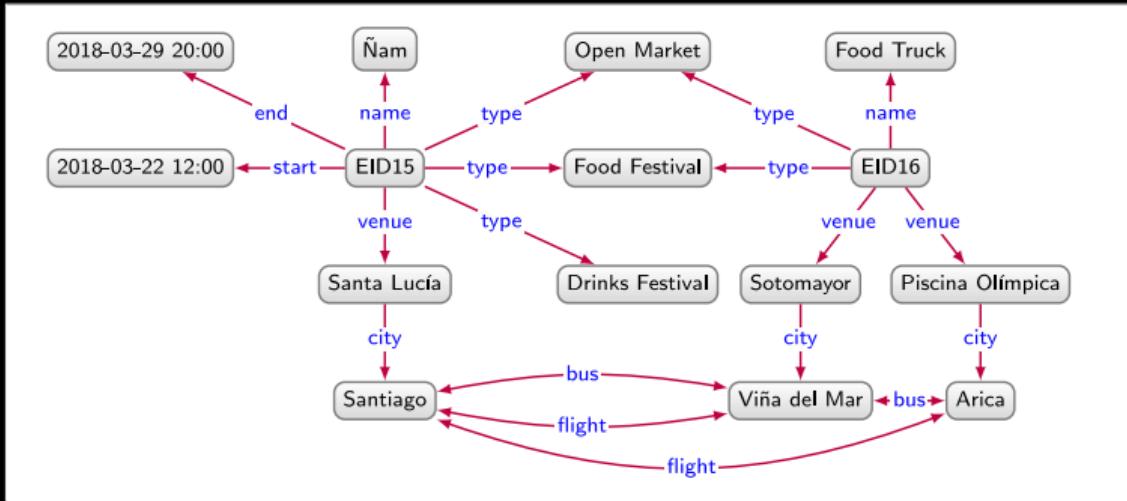
Definition

A **directed edge-labeled graph (DELG)** is a tuple $G = (V, E, L)$, where

- $V \subseteq \text{Con}$ is the set of **nodes**,
- $L \subseteq \text{Con}$ is the set of **edge labels**, and
- $E \subseteq V \times L \times V$ be the set of **labeled edges**, i.e., triples of the form (node, edge label, node).

Note that V and L need **not** be disjoint (unlike the typical graphs you learned in the Discrete Math classes).

DELG: Example



List all elements of V , L , and E .

DELG: Example

- $V = \{2018-03-29 \text{ 20:00}, \tilde{\text{Nam}}, \text{Open Market}, \text{Food Truck}, 2018-03-22 \text{ 12:00}, \text{EID15}, \text{Food Festival}, \text{EID16}, \text{Santa Lucía}, \text{Drinks Festival}, \text{Sotomayor}, \text{Piscina Olímpica}, \text{Santiago}, \text{Viña del Mar}, \text{Arica}\}$
- $L = \{\text{name, type, venue, start, end, city, bus, flight}\}$
- $E = \{(\text{EID15, name, } \tilde{\text{Nam}}), (\text{EID, start, } 2018-03-22 \text{ 12:00}), (\text{EID15, end, } 2018-03-29 \text{ 20:00})$
 $(\text{EID15, venue, Santa Lucía}), (\text{EID15, type, Open Market}), (\text{EID15, type, Food Festival}),$
 $(\text{EID15, type, Drinks Festival}), (\text{EID16, name, Food Truck}), (\text{EID16, type, Open Market}),$
 $(\text{EID16, type, Food Festival}), (\text{EID16, venue, Sotomayor}), (\text{EID16, venue, Piscina Olímpica}),$
 $(\text{Santa Lucía, city, Santiago}), (\text{Sotomayor, city, Viña del Mar}), (\text{Piscina Olímpica, city, Arica})$
 $(\text{Santiago, bus, Viña del Mar}), (\text{Viña del Mar, bus, Santiago}), (\text{Santiago, flight, Viña del Mar})$
 $(\text{Viña del Mar, flight, Santiago}), (\text{Santiago, flight, Arica}), (\text{Arica, flight, Santiago}),$
 $(\text{Viña del Mar, bus, Arica}), (\text{Arica, bus, Viña del Mar})\}$

DELG: Remarks

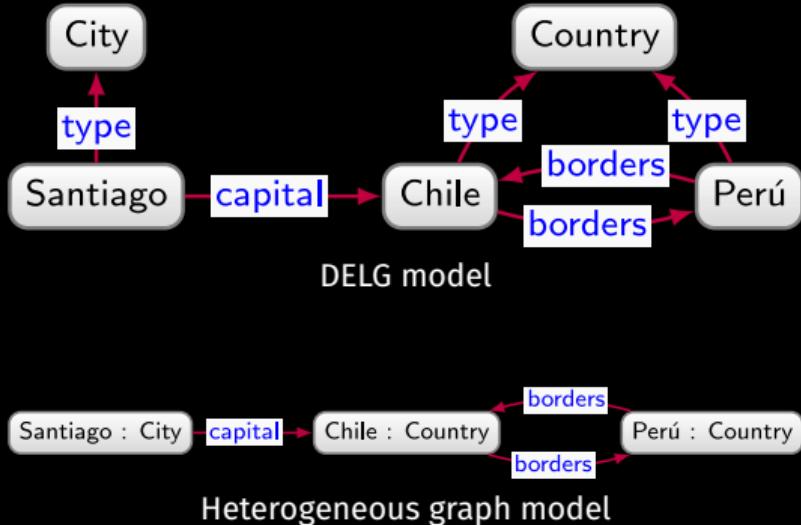
- We can treat the set of edges E as the DELG without explicitly referring to the set of nodes V and labels L .
 - Given a set of edges E , we define the **graph induced by E** as $G = (V, E, L)$ such that V comprises the nodes at either end of any edge in E and L comprises the labels of any edge in E .
- We can perform set operations on DELGs, which are understood as the application of those operations to their sets of edges.
 - Given DELG $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, we define the DELG $G_1 \cup G_2$ as the graph induced by $E_1 \cup E_2$
 - W3C standards for DELG data model is **Resource Description Framework (RDF)**.

Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

Heterogeneous graph: Example

Heterogeneous graphs are like DELGs, except that they treat the type edges as part of the graph model directly, instead of as a binary relation.



Heterogeneous graphs

Let Con be a countably infinite set of constants.

Definition

A **heterogeneous graph** is a tuple $G = (V, E, L, \ell)$, where

- $V \subseteq \text{Con}$ is a set of **nodes**,
- $L \subseteq \text{Con}$ is a set of **edge/node labels**,
- $E \subseteq V \times L \times V$ is a set of **labeled edges**, and
- $\ell: V \rightarrow L$ is a function that maps each node to a label.

Edge and node labels in heterogeneous graphs are called **types**.

Heterogeneous graph: Example



- $V = \{\text{Santiago, Chile, Perú}\}$
- $L = \{\text{City, Country}\}$
- $E = \{(\text{Santiago, capital, Chile}), (\text{Chile, borders, Perú}), (\text{Perú, borders, Chile})\}$
- $\ell = \{\text{Santiago} \mapsto \text{City}, \text{Chile} \mapsto \text{Country}, \text{Perú} \mapsto \text{Country}\}$

Heterogeneous graphs: Remarks

- An edge in a heterogeneous graph is called **homogeneous** if it is between two nodes of the same types/labels. Otherwise, it is called **heterogeneous**. (Which edges are homogeneous and which are heterogeneous in the previous example?)
- Heterogeneous graphs allow for the **partitioning of their nodes** according to their type – useful, e.g., for the purposes of machine learning tasks.
- Unlike DELG, heterogeneous graphs only support a **many-to-one relation between nodes and their types**, i.e., each node is associated exactly with one type.

Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

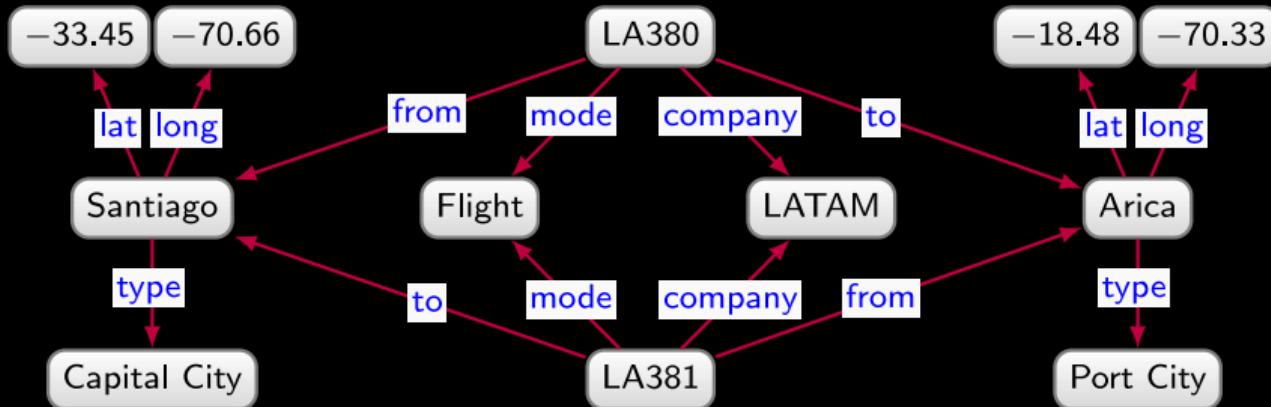
Motivation

- Consider the case where the tourism board would like to integrate incoming data that provide further details on which companies offer fares on which flights, hence allowing better understanding of available routes between cities.

Motivation

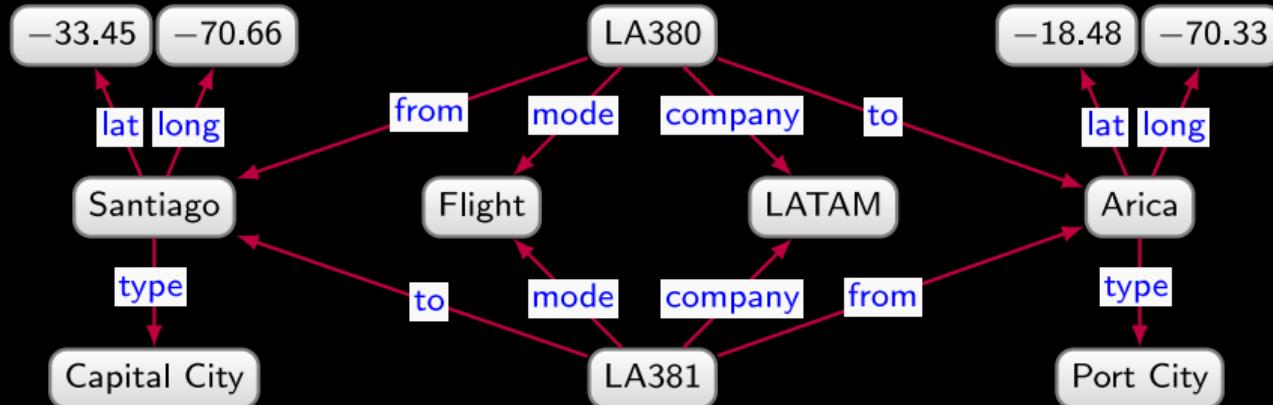
- Consider the case where the tourism board would like to integrate incoming data that provide further details on which companies offer fares on which flights, hence allowing better understanding of available routes between cities.
- Using DELG, we **cannot** directly annotate an edge like (Santiago, flight, Arica) with the companies offering that route.

Motivation



- In DELG, we could add a new node denoting a flight, then connect this node with the source, destination, companies, and mode.

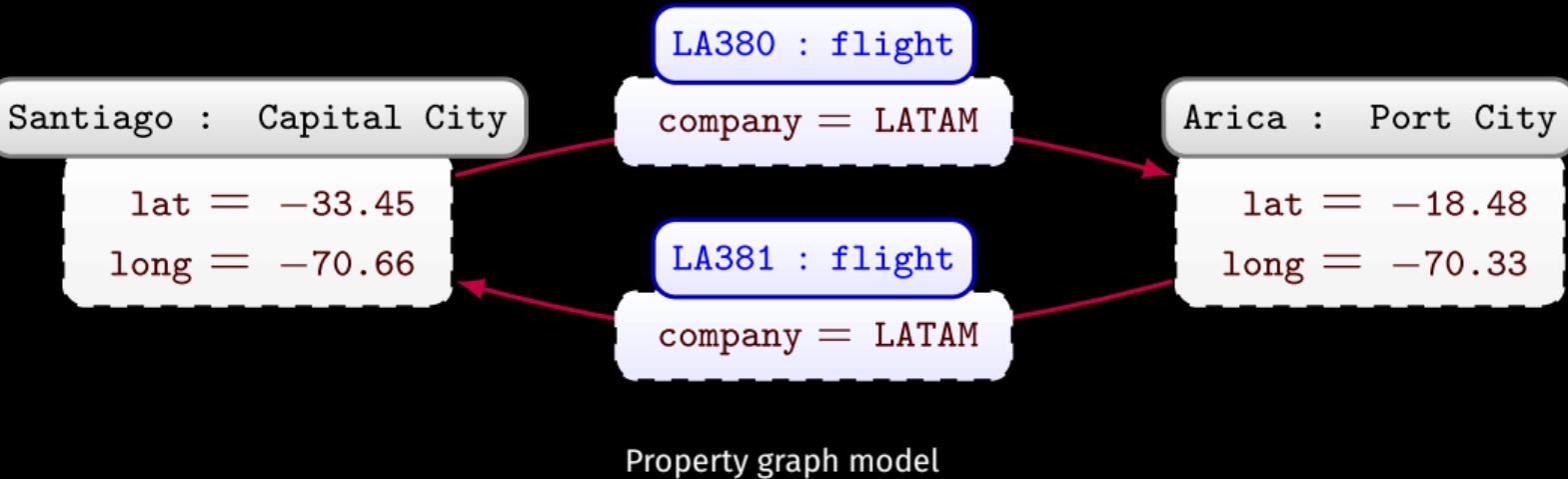
Motivation



- In DELG, we could add a new node denoting a flight, then connect this node with the source, destination, companies, and mode.
- However, applying this to all routes would involve significant changes.

Property graph: Example

As alternative to DELG, property graphs allow a set of **property-value** pairs to be associated with both nodes and edges, offering more flexibility.



Property graphs

Let Con be a countably infinite set of constants.

Definition

A **property graph** is a tuple $G = (V, E, L, P, U, e, \ell, p)$, where

- $V \subseteq \text{Con}$ is a set of **node ids**,
- $E \subseteq \text{Con}$ is a set of **edge ids**
- $L \subseteq \text{Con}$ is a set of **labels**,
- $P \subseteq \text{Con}$ is a set of **properties**,
- $U \subseteq \text{Con}$ is a set of **values**,
- $e: E \rightarrow V \times V$ is a function that maps an edge id to a pair of node ids,
- $\ell: V \cup E \rightarrow 2^L$ is a function that maps a node or edge id to a **set of labels**, and
- $p: V \cup E \rightarrow 2^{P \times U}$ is a function that maps a node or edge id to a **set of property-value pairs**.

Property graph: Example



List all elements of V, E, L, P, U, e, ℓ , and p

Property graph: Example

- $V = \{\text{Santiago, Arica}\}$
- $E = \{\text{LA380, LA381}\}$
- $L = \{\text{Capital City, Port City, flight}\}$
- $P = \{\text{lat, long, company}\}$
- $U = \{-33.45, -70.66, -18.48, -70.33, \text{LATAM}\}$
- $e = \{\text{LA380} \mapsto (\text{Santiago, Arica}), \text{LA381} \mapsto (\text{Arica, Santiago})\}$
- $\ell = \{\text{Santiago} \mapsto \{\text{Capital City}\}, \text{Arica} \mapsto \{\text{Port City}\}, \text{LA380} \mapsto \{\text{flight}\}, \text{LA381} \mapsto \{\text{flight}\}\}$
- $p = \{\text{Santiago} \mapsto \{(\text{lat}, -33.45), (\text{long}, -70.66)\}, \text{LA380} \mapsto \{(\text{company}, \text{LATAM})\} \\ \text{Arica} \mapsto \{(\text{lat}, -18.48), (\text{long}, -70.33)\}, \text{LA381} \mapsto \{(\text{company}, \text{LATAM})\} \}$

Property graph: Remarks

- Property graphs are prominently used in graph databases, such as in Neo4j.
- Property graphs can be converted to/from DELG.
- DELGs offers a more minimal model, while property graphs offer a more flexible one.
- Choice of models are also often dictated by other factors, e.g., the availability of implementation for different models.

Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)
- Two possibilities to handle multiple KGs:

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)
- Two possibilities to handle multiple KGs:
 - merge the graphs together into one monolithic graph by taking their union; or

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)
- Two possibilities to handle multiple KGs:
 - merge the graphs together into one monolithic graph by taking their union; or
 - manage a graph dataset containing multiple graphs that are separated from each other.

Graph datasets (contd.)

Let Con be a countably infinite set of constants.

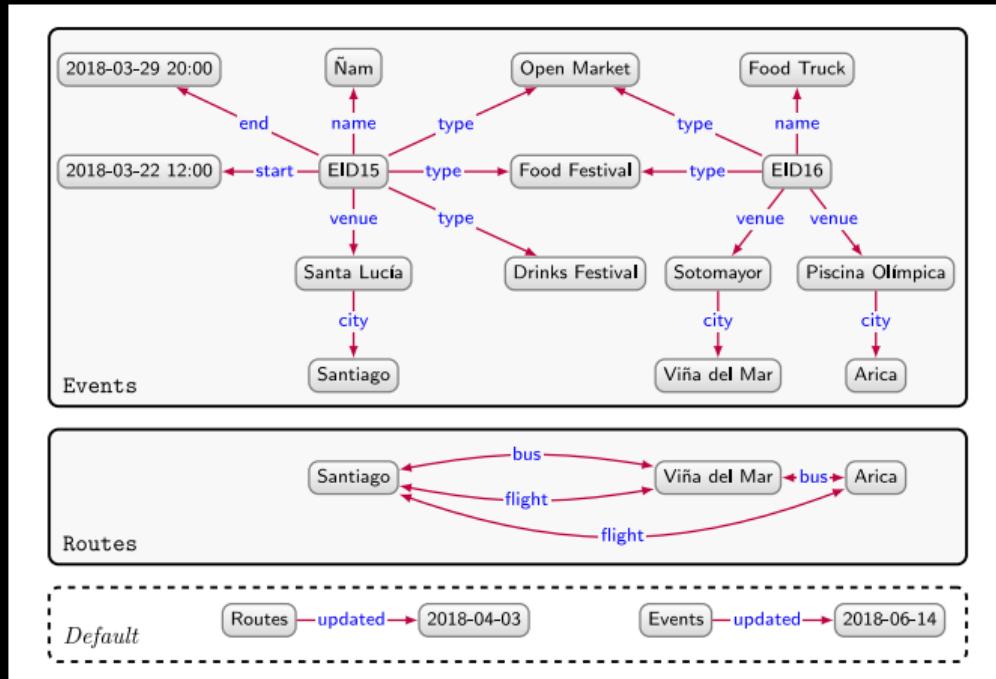
Definition

A **named graph** is a pair (n, G) where G is a data graph and $n \in \text{Con}$ is a graph name/ID.

A **graph dataset** is a tuple $\mathcal{G} = (G_D, \{(n_1, G_1), \dots, (n_k, G_k)\})$ with $k \geq 0$ where

- G_D is a data graph, called the **default graph**, which is allowed to be empty.
- (n_i, G_i) is a **named graph** for each i , $1 \leq i \leq k$
- if $k > 0$, then for each $1 \leq i, j \leq k$, whenever $i \neq j$, then $n_i \neq n_j$

Graph datasets: Example



Graph dataset with
a default graph and
two named graphs
(Routes and
Events)

Graph datasets: Remarks

- A graph dataset consists of exactly one default graph and zero or more named graphs.
- Named graphs are uniquely named, i.e., no two named graphs in the same graph dataset have the same name.
- The default graph of a graph dataset is the only graph without an ID in that dataset.
- Graph names can be used as nodes in any graph in the dataset.
- Nodes and edges may be repeated across different graphs in the dataset (and they refer to the same entity).
- Using a graph dataset allows us to:
 - update or refine data from different sources separately;
 - distinguish untrustworthy sources from the more trustworthy ones;
 - etc.

Outline

1. Knowledge Graph: An Overview
2. Data Graph
3. Directed Edge-labeled Graphs
4. Heterogeneous Graphs
5. Property Graphs
6. Graph Datasets
7. Other Graph Data Models and Graph Stores

Other graph data models

- Graph data models other than the three already discussed exist, e.g.,
 - graphs with complex nodes (called hypernodes), which may contain individual edges or nested graphs;
 - (hyper)graphs with complex edges that connect sets rather than pairs of nodes.
- Conversion between different graph data models is often possible.
- KG can adopt any of the aforementioned models, but we focus only on DELGs.

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
 - a single relation of arity three, i.e., a **triple table**; or

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
 - a single relation of arity three, i.e., a **triple table**; or
 - a collection of binary relations/tables, one for each property, i.e., **vertical partitioning**; or

Graph stores

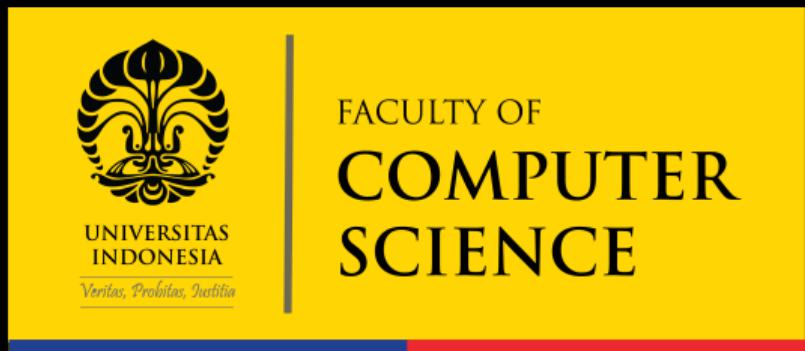
- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
 - a single relation of arity three, i.e., a **triple table**; or
 - a collection of binary relations/tables, one for each property, i.e., **vertical partitioning**; or
 - a collection of n -ary relations/tables, each contains entities of the same type, i.e., **property tables**.

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
 - a single relation of arity three, i.e., a **triple table**; or
 - a collection of binary relations/tables, one for each property, i.e., **vertical partitioning**; or
 - a collection of n -ary relations/tables, each contains entities of the same type, i.e., **property tables**.
- DELGs can also be stored using **custom/native storage techniques** (not using tables), which may provide more efficient access for finding nodes, edges, and their adjacent elements.

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
 - a single relation of arity three, i.e., a **triple table**; or
 - a collection of binary relations/tables, one for each property, i.e., **vertical partitioning**; or
 - a collection of n -ary relations/tables, each contains entities of the same type, i.e., **property tables**.
- DELGs can also be stored using **custom/native storage techniques** (not using tables), which may provide more efficient access for finding nodes, edges, and their adjacent elements.
- Some systems further allow distributing graphs over multiple machines based on NoSQL stores or custom partitioning schemes.



Semantic Web 03: RDF and the Issue of Identity

Adila Krisnadhi – adila@cs.ui.ac.id

Faculty of Computer Science, Universitas

Indonesia

Outline

1. Persistent Identifiers
2. Datatypes
3. Resource Description Framework (RDF)
4. Existential Nodes
5. Lexicalization

Persistent identifiers: Motivation

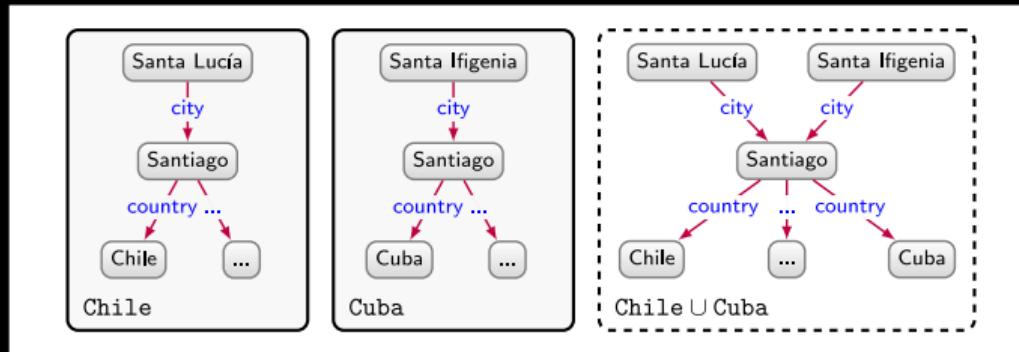
- Suppose we obtain two different graphs: one describes tourism in Chile, while the other describes tourism in Cuba.
- We can merge both graphs together, but ...

Persistent identifiers: Motivation

- Suppose we obtain two different graphs: one describes tourism in Chile, while the other describes tourism in Cuba.
- We can merge both graphs together, but ... there might be a problem: both Chile and Cuba have a city named Santiago!

Persistent identifiers: Motivation

- Suppose we obtain two different graphs: one describes tourism in Chile, while the other describes tourism in Cuba.
- We can merge both graphs together, but ... there might be a problem: both Chile and Cuba have a city named Santiago!



Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.

Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.
- Solution: use long lasting, **globally unique**, **persistent identifiers** (PIPs).

Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.
- Solution: use long lasting, **globally unique**, **persistent identifiers** (PIPs).
 - Digital Object Identifiers (**DOIs**) for papers,

Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.
- Solution: use long lasting, **globally unique**, **persistent identifiers** (PIPs).
 - Digital Object Identifiers (**DOIs**) for papers,
 - ORCID IDs for researchers,

Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.
- Solution: use long lasting, **globally unique, persistent identifiers** (PIPs).
 - Digital Object Identifiers (DOIs) for papers,
 - ORCID IDs for researchers,
 - International Standard Book Numbers (ISBNs) for books,

Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.
- Solution: use long lasting, **globally unique, persistent identifiers** (PIPs).
 - Digital Object Identifiers (DOIs) for papers,
 - ORCID IDs for researchers,
 - International Standard Book Numbers (ISBNs) for books,
 - Alpha-2 codes for countries,

Persistent identifiers: Motivation

- Using an **ambiguous** node may lead to **naming clash**.
- Solution: use long lasting, **globally unique, persistent identifiers** (PIPs).
 - Digital Object Identifiers (DOIs) for papers,
 - ORCID IDs for researchers,
 - International Standard Book Numbers (ISBNs) for books,
 - Alpha-2 codes for countries,
 - for Semantic Web resources?

Can we use URL?

- For the Web, we already have **Uniform Resource Locators (URLs)**:
 - identifies the location of **information resources** (i.e., web documents) such as webpages.

Can we use URL?

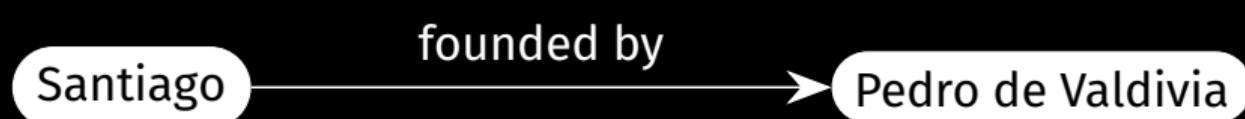
- For the Web, we already have **Uniform Resource Locators (URLs)**:
 - identifies the location of **information resources** (i.e., web documents) such as webpages.
- However, URLs are not sufficient ...

Can we use URL?

- For the Web, we already have **Uniform Resource Locators (URLs)**:
 - identifies the location of **information resources** (i.e., web documents) such as webpages.
- However, URLs are not sufficient ...
- Consider as an example, we would like to represent/store the information that “Santiago in Chile was founded by Pedro de Valdivia”, leading to the following graph:

Can we use URL?

- For the Web, we already have **Uniform Resource Locators (URLs)**:
 - identifies the location of **information resources** (i.e., web documents) such as webpages.
- However, URLs are not sufficient ...
- Consider as an example, we would like to represent/store the information that “Santiago in Chile was founded by Pedro de Valdivia”, leading to the following graph:



Santiago was founded by Pedro de Valdivia

Suppose the relevant URLs are as follows:

- URL for Santiago of Chile's **webpage** in Wikidata:
<https://www.wikidata.org/wiki/Q2887>
- URL for Pedro de Valdivia's **webpage** in Wikidata:
<https://www.wikidata.org/wiki/Q203534>
- URL for the **webpage** of the 'founded by' relation in Wikidata:
<https://www.wikidata.org/wiki/Property:P112>

Then we have something like:

Santiago was founded by Pedro de Valdivia

Suppose the relevant URLs are as follows:

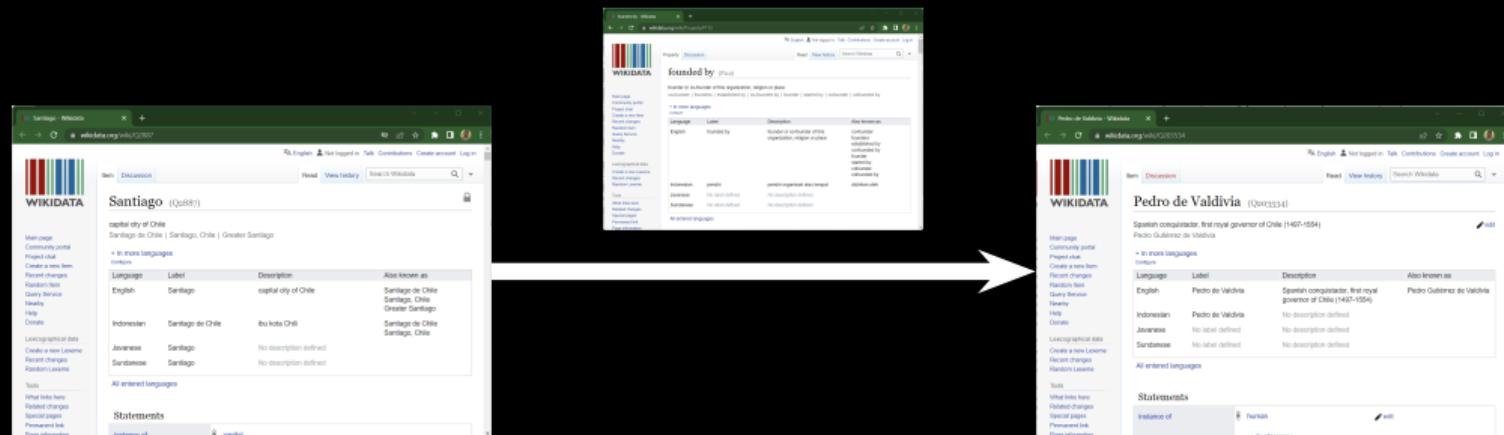
- URL for Santiago of Chile's **webpage** in Wikidata:
<https://www.wikidata.org/wiki/Q2887>
- URL for Pedro de Valdivia's **webpage** in Wikidata:
<https://www.wikidata.org/wiki/Q203534>
- URL for the **webpage** of the 'founded by' relation in Wikidata:
<https://www.wikidata.org/wiki/Property:P112>

Then we have something like:



Santiago was founded by Pedro de Valdivia (cont.)

which actually represents the following relationship **between the two webpages** (not between a city and a person):



Santiago was founded by Pedro de Valdivia (cont.)

Santiago - Wikidata x +

wikidata.org/wiki/Q2887

English Not logged in Talk Contributions Create account Log in

Item Discussion Read View history Search Wikidata

Santiago (Q2887)

capital city of Chile

Santiago de Chile | Santiago, Chile | Greater Santiago

In more languages Configure

Language	Label	Description	Also known as
English	Santiago	capital city of Chile	Santiago de Chile Santiago, Chile Greater Santiago
Indonesian	Santiago de Chile	Ibu kota Chili	Santiago de Chile Santiago, Chile
Javanese	Santiago	No description defined	
Sundanese	Santiago	No description defined	

All entered languages

Statements

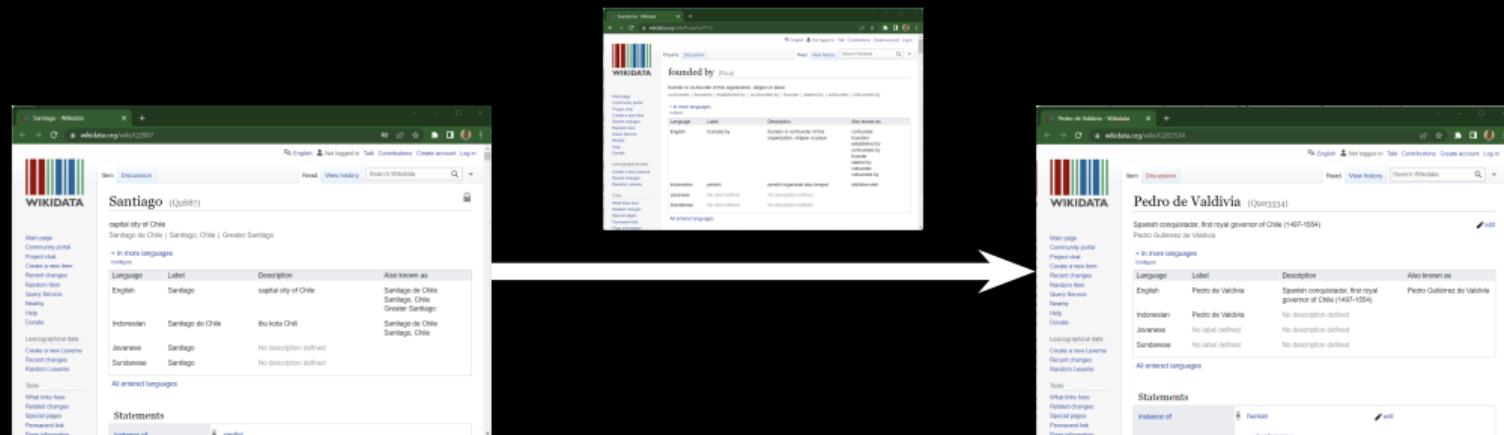
instance of capital

In more languages		
Configure		
Language	Label	Description
English	founded by	founder or co-founder of organization, re-
Indonesian	pendiri	pendiri organisasi
Javanese	No label defined	No description
Sundanese	No label defined	No description

All entered languages

Santiago was founded by Pedro de Valdivia (cont.)

which actually represents the following relationship **between the two webpages** (not between a city and a person):



Santiago was founded by Pedro de Valdivia (cont.)



Pedro de Valdivia - Wikidata

wikidata.org/wiki/Q203534

Item Discussion Read View history Search Wikidata

Pedro de Valdivia (Q203534)

Spanish conquistador, first royal governor of Chile (1497-1554) [edit](#)

Pedro Gutiérrez de Valdivia

In more languages [Configure](#)

Language	Label	Description	Also known as
English	Pedro de Valdivia	Spanish conquistador, first royal governor of Chile (1497-1554)	Pedro Gutiérrez de Valdivia
Indonesian	Pedro de Valdivia	No description defined	
Javanese	No label defined	No description defined	
Sundanese	No label defined	No description defined	

All entered languages

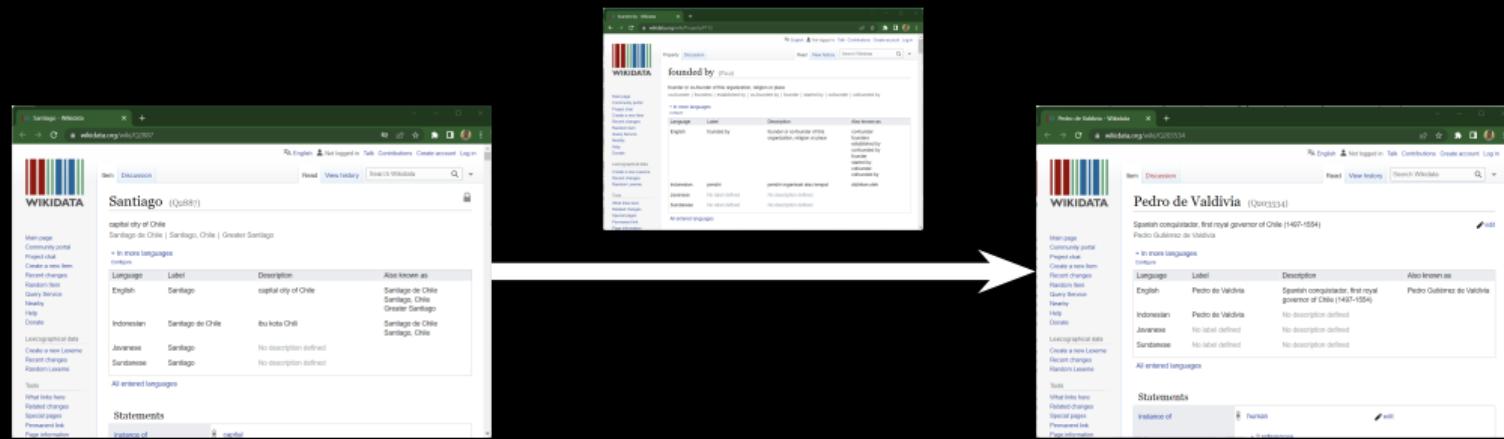
Statements

instance of	human	edit
-------------	-------	----------------------

↳ 2 references

Santiago was founded by Pedro de Valdivia (cont.)

which actually represents the following relationship **between the two webpages** (not between a city and a person):



The image shows three screenshots of Wikidata pages arranged horizontally, connected by a large white arrow pointing from left to right.

- Left Page (Santiago - Wikipedia):** Shows the statement "Santiago was founded by Pedro de Valdivia".
- Middle Page (Pedro de Valdivia - Wikipedia):** Shows the statement "Pedro de Valdivia founded Santiago".
- Right Page (Pedro de Valdivia - Wikipedia):** Shows the statement "Pedro de Valdivia was born in Santiago".

This illustrates how the same relationships can be represented in different ways across different entities, leading to potential ambiguity in RDF graphs.

- What does it mean to have a webpage as a relation?
- Relationship is ambiguous: was Pedro de Valdivia the founder of the webpage or the city?

- For the Semantic Web, we extend the idea of URLs to the so-called Internationalized Resource Identifier (IRI) – used to be called Uniform Resource Identifier (URI).

- For the Semantic Web, we extend the idea of URLs to the so-called Internationalized Resource Identifier (IRI) – used to be called Uniform Resource Identifier (URI).
- An IRI is first and foremost an identifier, but in many cases, it is also a location on the Web.

- For the Semantic Web, we extend the idea of URLs to the so-called **Internationalized Resource Identifier (IRI)** – used to be called **Uniform Resource Identifier (URI)**.
- An IRI is first and foremost an identifier, but in many cases, it is also a location on the Web.
- IRIs look similar to URLs; in fact, every URL is an IRI, because every URL **is** an identifier of some web document.

- For the Semantic Web, we extend the idea of URLs to the so-called **Internationalized Resource Identifier (IRI)** – used to be called **Uniform Resource Identifier (URI)**.
- An IRI is first and foremost an identifier, but in many cases, it is also a location on the Web.
- IRIs look similar to URLs; in fact, every URL is an IRI, because every URL **is** an identifier of some web document.
- IRIs also include identifiers assigned to **non-information resources**, e.g., people, cities, organizations, etc. For example,

- For the Semantic Web, we extend the idea of URLs to the so-called **Internationalized Resource Identifier (IRI)** – used to be called **Uniform Resource Identifier (URI)**.
- An IRI is first and foremost an identifier, but in many cases, it is also a location on the Web.
- IRIs look similar to URLs; in fact, every URL is an IRI, because every URL **is** an identifier of some web document.
- IRIs also include identifiers assigned to **non-information resources**, e.g., people, cities, organizations, etc. For example,
 - For Santiago the city, Wikidata uses <https://www.wikidata.org/entity/Q2887>

- For the Semantic Web, we extend the idea of URLs to the so-called **Internationalized Resource Identifier (IRI)** – used to be called **Uniform Resource Identifier (URI)**.
- An IRI is first and foremost an identifier, but in many cases, it is also a location on the Web.
- IRIs look similar to URLs; in fact, every URL is an IRI, because every URL **is** an identifier of some web document.
- IRIs also include identifiers assigned to **non-information resources**, e.g., people, cities, organizations, etc. For example,
 - For Santiago the city, Wikidata uses <https://www.wikidata.org/entity/Q2887>
 - For Pedro de Valdivia the person, Wikidata uses <https://www.wikidata.org/wiki/Q203534>

- For the Semantic Web, we extend the idea of URLs to the so-called **Internationalized Resource Identifier (IRI)** – used to be called **Uniform Resource Identifier (URI)**.
- An IRI is first and foremost an identifier, but in many cases, it is also a location on the Web.
- IRIs look similar to URLs; in fact, every URL is an IRI, because every URL **is** an identifier of some web document.
- IRIs also include identifiers assigned to **non-information resources**, e.g., people, cities, organizations, etc. For example,
 - For Santiago the city, Wikidata uses <https://www.wikidata.org/entity/Q2887>
 - For Pedro de Valdivia the person, Wikidata uses <https://www.wikidata.org/wiki/Q203534>
 - For the ‘founded by’ relationship, Wikidata uses <https://www.wikidata.org/prop/direct/P112>

Santiago was founded by Pedro de Valdivia with IRIs

The relationship between Santiago the city and Pedro de Valdivia is represented by:



IRI namespace

- The IRI scheme used to distinguish information and non-information resources is left to the information/data owner/maintainer so long as it conforms to the general scheme for IRI (see next slide).

IRI namespace

- The IRI scheme used to distinguish information and non-information resources is left to the information/data owner/maintainer so long as it conforms to the general scheme for IRI (see next slide).
- The prefix part of each IRI is known as a **namespace**, and we can define an abbreviation for them, e.g.,

IRI namespace

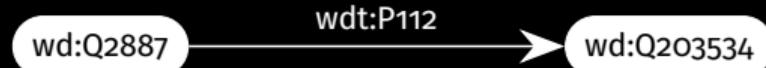
- The IRI scheme used to distinguish information and non-information resources is left to the information/data owner/maintainer so long as it conforms to the general scheme for IRI (see next slide).
- The prefix part of each IRI is known as a **namespace**, and we can define an abbreviation for them, e.g.,
 - <https://www.wikidata.org/entity/> is abbreviated as wd:

IRI namespace

- The IRI scheme used to distinguish information and non-information resources is left to the information/data owner/maintainer so long as it conforms to the general scheme for IRI (see next slide).
- The prefix part of each IRI is known as a **namespace**, and we can define an abbreviation for them, e.g.,
 - `https://www.wikidata.org/entity/` is abbreviated as `wd:`
 - `https://www.wikidata.org/prop/direct/` is abbreviated as `wdt:`

IRI namespace

- The IRI scheme used to distinguish information and non-information resources is left to the information/data owner/maintainer so long as it conforms to the general scheme for IRI (see next slide).
- The prefix part of each IRI is known as a **namespace**, and we can define an abbreviation for them, e.g.,
 - `https://www.wikidata.org/entity/` is abbreviated as `wd:`
 - `https://www.wikidata.org/prop/direct/` is abbreviated as `wdt:`
- So, “Santiago was founded by Pedro Valdivia” can be represented by:



IRI standard according to RFC 3987

IRI ::= *scheme*: [// *authority*] *path* [? *query*] [# *fragment*]

- All parts outside ‘:’, ‘/’, ‘?’, and ‘#’ may use Unicode characters.
- Parts in square brackets are optional.
- **scheme**: the scheme classifying the type of IRI
 - E.g., http, https, ftp, mailto, urn, ...
 - Standardized by IANA (Internet Assigned Numbers Authority)
 - Case insensitive if using US-ASCII
- **authority**: host name, optionally with user and/or port info
 - E.g., kgbook.org, john@example.com, example.org:8080
 - host names using US-ASCII are case-insensitive.

IRI standard according to RFC 3987 (cont.)

- **path**: main part of IRIs organized hierarchically
 - E.g., /etc/passwd, this/path/with/-:_ /is/..okay
 - Case-sensitive, if using US-ASCII, and may be empty (e.g., in email address)
- **query**: optional part of the IRI that provides additional non-hierarchical information such as providing HTTP GET parameters
 - E.g., q=Semantic+Web+book
 - Case-sensitive if using US-ASCII
- **fragment**: provides a second level of identifying resources (different fragments mean different names even if they lead to the same document when retrieved in browser)
 - E.g., #section1
 - Case-sensitive if using US-ASCII

IRI examples

- <https://en.wikipedia.org/wiki/Indonesia#History>
- <https://remote-lib.ui.ac.id:2196/doi/fullHtml/10.1145/3293318>
- <https://arxiv.org/pdf/1806.06478.pdf>
- <https://www.google.com/search?client=opera&q=knowledge+graph+zero+shot+learning&sourceid=opera&ie=UTF-8&oe=UTF-8>
- <mailto:adila@cs.ui.ac.id>

IRI as identifier

- Using http or https for the scheme is preferred, because http/https IRIs can be resolved via the Web protocol, i.e., the HTTP protocol.
- If two IRIs are the same in all parts except that one uses http and the other uses https, then they are considered the same.
- IRIs should be **persistent**:
 - should always be live and point to the same resource forever
 - the query part should be empty.
- Persistent URL (PURL) services may be used to ensure persistence.
 - They offer redirects from a fixed central server to a particular location, which may be changed over time.
 - See <http://www.purlz.org/> and <https://w3id.org/> for more details

IRI is globally unique?

- The scheme part is standardized and a data owner picks one for the host machine in which the data would be hosted.

IRI is globally unique?

- The scheme part is standardized and a data owner picks one for the host machine in which the data would be hosted.
- The authority part, i.e., host name corresponds to a particular data owner (person, organization, etc.) who holds the only authority over the host machine (i.e., no other data owner in the world holds the authority for that machine).

IRI is globally unique?

- The scheme part is standardized and a data owner picks one for the host machine in which the data would be hosted.
- The authority part, i.e., host name corresponds to a particular data owner (person, organization, etc.) who holds the only authority over the host machine (i.e., no other data owner in the world holds the authority for that machine).
- The data owner also holds the only authority on how the rest of the IRI is defined.

IRI is globally unique?

- The scheme part is standardized and a data owner picks one for the host machine in which the data would be hosted.
- The authority part, i.e., host name corresponds to a particular data owner (person, organization, etc.) who holds the only authority over the host machine (i.e., no other data owner in the world holds the authority for that machine).
- The data owner also holds the only authority on how the rest of the IRI is defined.
 - Uniqueness is ensured if the data owner ensures that each resource (entity or relation) is assigned a unique combination of path (and fragment if employed).

IRI is globally unique?

- The scheme part is standardized and a data owner picks one for the host machine in which the data would be hosted.
- The authority part, i.e., host name corresponds to a particular data owner (person, organization, etc.) who holds the only authority over the host machine (i.e., no other data owner in the world holds the authority for that machine).
- The data owner also holds the only authority on how the rest of the IRI is defined.
 - Uniqueness is ensured if the data owner ensures that each resource (entity or relation) is assigned a unique combination of path (and fragment if employed).
- Hence, an IRI is globally unique: each IRI corresponds to exactly one resource (entity/relation).

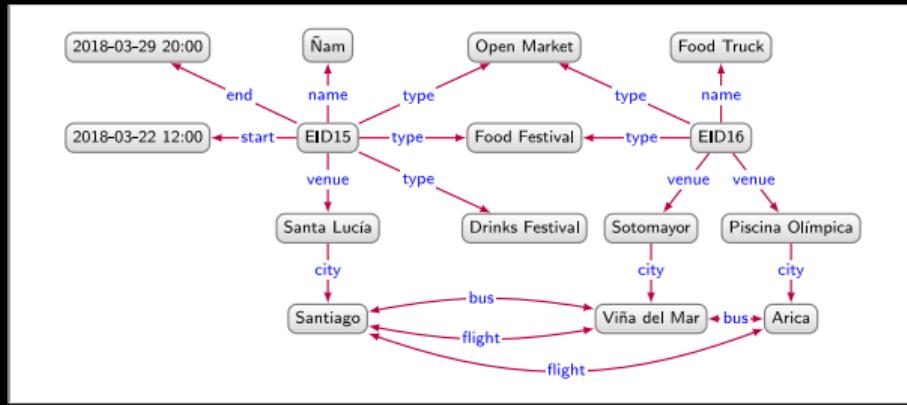
IRI is globally unique?

- The scheme part is standardized and a data owner picks one for the host machine in which the data would be hosted.
- The authority part, i.e., host name corresponds to a particular data owner (person, organization, etc.) who holds the only authority over the host machine (i.e., no other data owner in the world holds the authority for that machine).
- The data owner also holds the only authority on how the rest of the IRI is defined.
 - Uniqueness is ensured if the data owner ensures that each resource (entity or relation) is assigned a unique combination of path (and fragment if employed).
- Hence, an IRI is globally unique: each IRI corresponds to exactly one resource (entity/relation).
 - But, this does not prevent two IRIs to refer to the same resource.

Outline

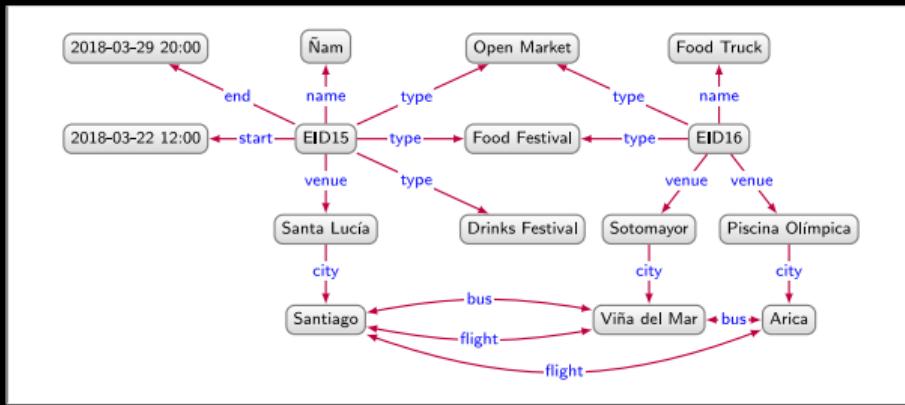
1. Persistent Identifiers
2. Datatypes
3. Resource Description Framework (RDF)
4. Existential Nodes
5. Lexicalization

Motivation



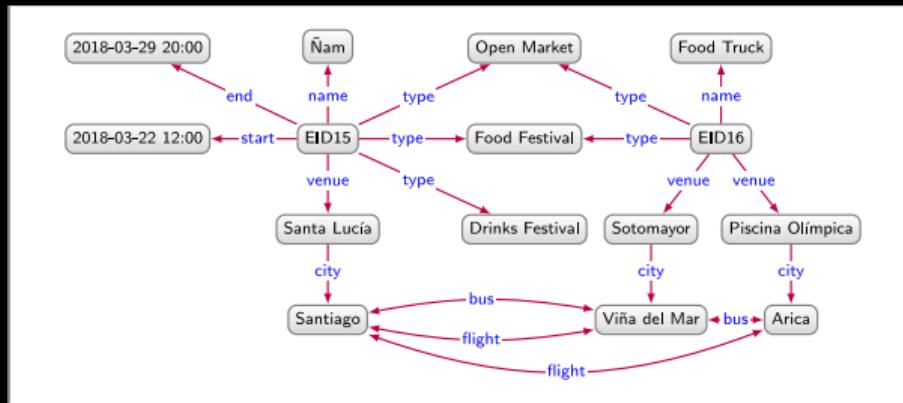
How should we assign a global/persistent identifier for the two date-times on the left?

Motivation



How should we assign a global/persistent identifier for the two date-times on the left?
Assigning IRIs doesn't make sense because their syntactic form:

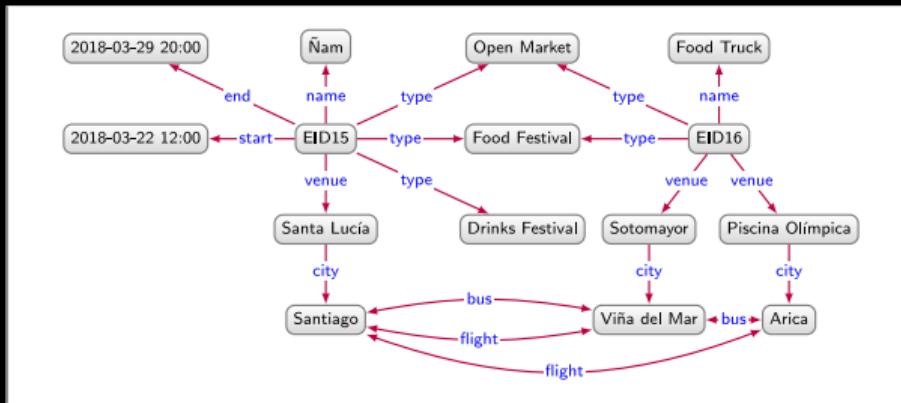
Motivation



How should we assign a global/persistent identifier for the two date-times on the left?
Assigning IRIs doesn't make sense because their syntactic form:

- directly indicates what they refer to: specific date and times in March 2020.

Motivation



How should we assign a global/persistent identifier for the two date-times on the left?
Assigning IRIs doesn't make sense because their syntactic form:

- directly indicates what they refer to: specific date and times in March 2020.
- is recognizable by machines – with appropriate software, we could sort them, extract the year, etc.

Datatypes

- Solution: take the node's value as a **literal** with a certain **datatype**.

Datatypes

- Solution: take the node's value as a **literal** with a certain **datatype**.
 - e.g., RDF writes the node (2018-03-22 12:00) as a literal string with a datatype
“2018-03-22T12:00:00”^^xsd:dateTime
- In RDF, such a node is **not** allowed to have an outgoing edge.

Datatypes

- Solution: take the node's value as a **literal** with a certain **datatype**.
 - e.g., RDF writes the node (2018-03-22 12:00) as a literal string with a datatype
“2018-03-22T12:00:00”^^xsd:dateTIme

In RDF, such a node is **not** allowed to have an outgoing edge.
 - xsd:dateTIme is an IRI denoting the datatype. Other IRI datatypes include xsd:string, xsd:decimal, etc.

Datatypes

- Solution: take the node's value as a **literal** with a certain **datatype**.
 - e.g., RDF writes the node (2018-03-22 12:00) as a literal string with a datatype
“2018-03-22T12:00:00”^^xsd:dateTIme

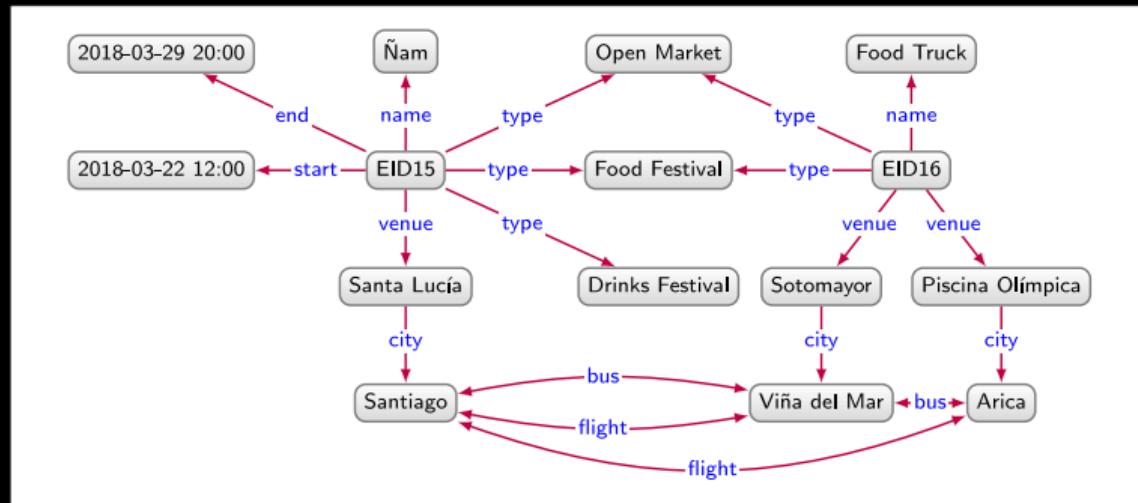
In RDF, such a node is **not** allowed to have an outgoing edge.
 - xsd:dateTIme is an IRI denoting the datatype. Other IRI datatypes include xsd:string, xsd:decimal, etc.
- Similar notions are also used in property graphs (though maybe not using IRIs).

Outline

1. Persistent Identifiers
2. Datatypes
3. Resource Description Framework (RDF)
4. Existential Nodes
5. Lexicalization

- **Resource Description Framework (RDF)**: W3C-standardized DELG data model.
 - RDF 1.0 in 2004
 - RDF 1.1 in 2014 (see <https://www.w3.org/TR/rdf11-primer/>)
- An **RDF triple** is a statement of the form (s, p, o) where
 - s , called the **subject** of the triple, is either a IRI or a blank node;
 - p , called the **predicate** of the triple, is an IRI; and
 - o , called the **object** of the triple, is an IRI or a blank node or a literal.
The literal can be assigned a datatype explicitly. Otherwise, the literal is typed xsd:string by default if not given explicitly.
- An **RDF graph** is a set of RDF triples.

Graph example



To model the above graph in RDF, we need to decide which nodes are literal (and their type) and which should be given an IRI.

Which nodes are literal?

- Nodes containing “basic/primitive” data values should be modeled as literals, e.g., strings, numbers, booleans, etc.
- In the previous examples, 4 nodes are literals:
 - two dates (given xsd:dateType as type):

"2018-03-29T20:00:00"^^xsd:dateType

"2018-03-22T12:00:00"^^xsd:dateType

- two names (can be given type xsd:string):

"Food Truck"

"Food Truck"^^xsd:string

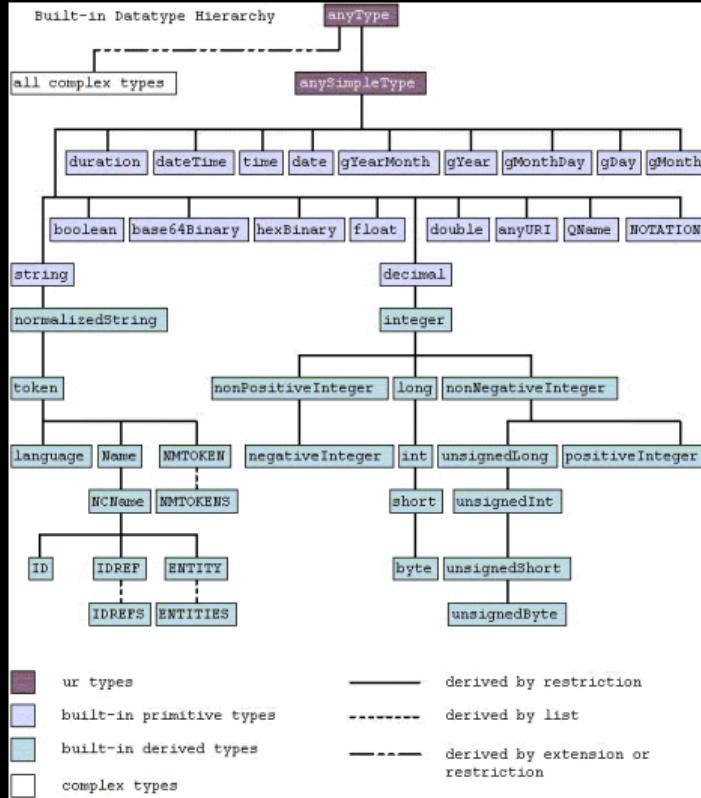
"Nam"

"Nam"^^xsd:string

How to give an IRI

- Pick one or more namespaces for your resources.
- Common practice is to set up different IRI patterns for relations, types, and the rest of the entities.
- For some nodes/relations/datatypes, if possible, it's better to reuse standard vocabulary terms. Some vocabulary terms have a standardized/community-agreed semantic that may be useful.
 - `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` or usually abbreviated `rdf:type` for the 'type' relation.
 - Many datatypes are defined as standard in the XML schema namespace, e.g.,
`http://www.w3.org/2001/XMLSchema#decimal` abbreviated
`xsd:decimal`
`http://www.w3.org/2001/XMLSchema#string` abbreviated `xsd:string`
 - See `http://prefix.cc` to obtain some well-known namespace IRIs.

XML schema type hierarchy

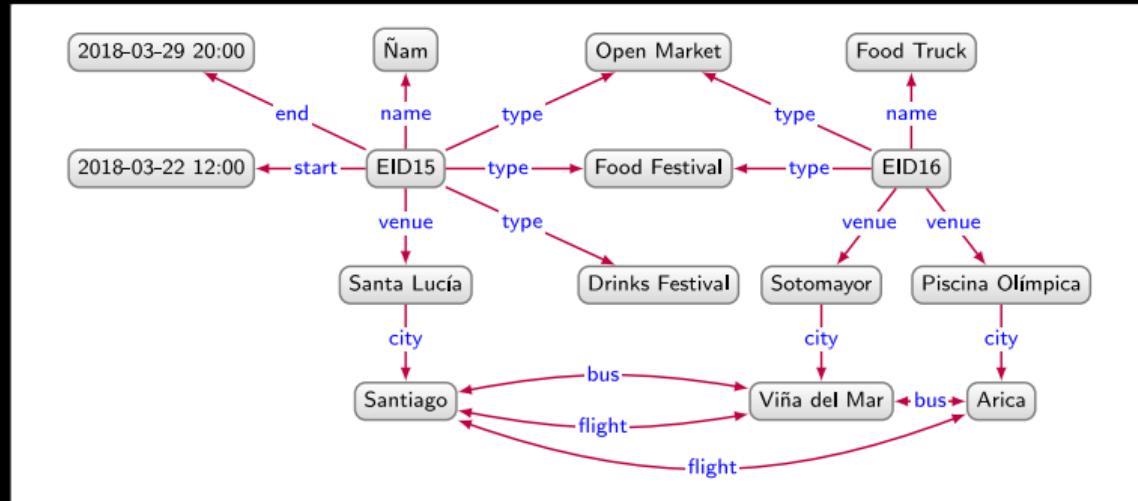


How to give an IRI (2)

For our example,

- we use the following IRI namespaces:
 - For instance nodes: `http://example.org/data/` abbreviated with `ex:`
 - For type nodes and relations (other than the 'type' relation):
`http://example.org/vocab#` abbreviated with `exv:`
- We simply take the node and property IDs in the graph to complete their identifier; white spaces are omitted.

RDF graph example



So for the graph above, we can write an RDF graph using N-triples, Turtle, RDF/XML, and JSON-LD in the following.

RDF graph example in N-triples

N-triples format:

- List all triples one by one in arbitrary order.
- Each triple ends with a period.
- All IRIs (including datatype IRIs) are written in full.
- Unicode characters are by default written in an escape sequence form.
- Typical file extension: .nt

RDF graph example in N-triples (cont.)

```
<http://example.org/data/Vi\u00F1adelMar> <http://example.org/vocab#bus>
  <http://example.org/data/Arica> .
<http://example.org/data/EID15> <http://example.org/vocab#venue>
  <http://example.org/data/SantaLuc\u00EDa> .
<http://example.org/data/SantaLuc\u00EDa> <http://example.org/vocab#city>
  <http://example.org/data/Santiago> .
<http://example.org/data/Arica> <http://example.org/vocab#bus>
  <http://example.org/data/Vi\u00F1adelMar> .
<http://example.org/data/Sotomayor> <http://example.org/vocab#city>
  <http://example.org/data/Vi\u00F1adelMar> .
<http://example.org/data/EID16> <http://example.org/vocab#venue>
  <http://example.org/data/Sotomayor> .
<http://example.org/data/EID16> <http://example.org/vocab#name>
  "Food Truck"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://example.org/data/PiscinaOl\u00EDmpica>
  <http://example.org/vocab#city> <http://example.org/data/Arica> .
<http://example.org/data/Santiago> <http://example.org/vocab#flight>
  <http://example.org/data/Vi\u00F1adelMar> .
```

RDF graph example in N-triples (cont.)

```
<http://example.org/data/EID16> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.org/vocab#FoodFestival> .
<http://example.org/data/EID15> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.org/vocab#FoodFestival> .
<http://example.org/data/EID15> <http://example.org/vocab#name>
  "\u00d1am" .
<http://example.org/data/EID16> <http://example.org/vocab#venue>
  <http://example.org/data/PiscinaOl\u00edmpica> .
<http://example.org/data/Vi\u00f1adelMar> <http://example.org/vocab#bus>
  <http://example.org/data/Santiago> .
<http://example.org/data/Arica> <http://example.org/vocab#flight>
  <http://example.org/data/Santiago> .
<http://example.org/data/EID15> <http://example.org/vocab#end>
  "2018-03-29T20:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://example.org/data/Santiago> <http://example.org/vocab#bus>
  <http://example.org/data/Vi\u00f1adelMar> .
<http://example.org/data/EID15> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.org/vocab#DrinksFestival> .
```

RDF graph example in N-triples (cont.)

```
<http://example.org/data/EID16> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.org/vocab#OpenMarket> .
<http://example.org/data/Santiago> <http://example.org/vocab#flight>
  <http://example.org/data/Arica> .
<http://example.org/data/EID15> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://example.org/vocab#OpenMarket> .
<http://example.org/data/Vi\u00f1adelMar> <http://example.org/vocab#flight>
  <http://example.org/data/Santiago> .
<http://example.org/data/EID15> <http://example.org/vocab#start>
  "2018-03-22T12:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

RDF graph example in Turtle

Turtle format:

- Triples (and triple groups) are ended by a period.
- Triples sharing the same subject can be grouped using semicolons.
- Triples sharing the same subject and predicate can be grouped using commas.
- Namespace prefixes can be used; defined using special syntax.
- Unicode characters need not be escaped.
- Typical file extension: .ttl
- Format conversion can be done using online tools, e.g.,
<https://issemantic.net/rdf-converter> (also has RDF visualizer) or
<https://www.easyrdf.org/converter>. Libraries such as Jena also supports it

RDF graph example in Turtle (cont.)

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix ex: <http://example.org/data/> .  
@prefix exv: <http://example.org/vocab#> .  
  
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival, exv:DrinksFestival ;  
    exv:name "Ñam" ;  
    exv:start "2018-03-22T12:00:00"^^xsd:dateTime ;  
    exv:end "2018-03-29T20:00:00"^^xsd:dateTime ;  
    exv:venue ex:SantaLucía .  
  
ex:SantaLucía exv:city ex:Santiago .  
  
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;  
    exv:name "Food Truck"^^xsd:string ;  
    exv:venue ex:Sotomayor, ex:PiscinaOlímpica .
```

RDF graph example in Turtle (cont.)

```
ex:Sotomayor exv:city ex:ViñadelMar .  
ex:PiscinaOlímpica exv:city ex:Arica .  
  
ex:Santiago exv:bus ex:ViñadelMar ;  
    exv:flight ex:ViñadelMar, ex:Arica .  
  
ex:ViñadelMar exv:bus ex:Santiago, ex:Arica ;  
    exv:flight ex:Santiago .  
  
ex:Arica exv:bus ex:ViñadelMar ;  
    exv:flight ex:Santiago .
```

RDF graph example in RDF/XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:exv="http://example.org/vocab#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="http://example.org/data/EID15">
    <rdf:type rdf:resource="http://example.org/vocab#OpenMarket"/>
    <exv:start rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2018-03-22T12:00:00</exv:start>
    <exv:name>Ñam</exv:name>
    <exv:venue rdf:resource="http://example.org/data/SantaLucía"/>
    <exv:end rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
      2018-03-29T20:00:00</exv:end>
    <rdf:type rdf:resource="http://example.org/vocab#DrinksFestival"/>
    <rdf:type rdf:resource="http://example.org/vocab#FoodFestival"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/data/ViñadelMar">
```

RDF graph example in RDF/XML (cont.)

```
<exv:bus rdf:resource="http://example.org/data/Santiago"/>
<exv:flight rdf:resource="http://example.org/data/Santiago"/>
<exv:bus rdf:resource="http://example.org/data/Arica"/>
</rdf:Description>
<rdf:Description rdf:about="http://example.org/data/Arica">
  <exv:bus rdf:resource="http://example.org/data/ViñadelMar"/>
  <exv:flight rdf:resource="http://example.org/data/Santiago"/>
</rdf:Description>
<rdf:Description rdf:about="http://example.org/data/EID16">
  <rdf:type rdf:resource="http://example.org/vocab#FoodFestival"/>
  <exv:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Food Truck</exv:name>
  <exv:venue rdf:resource="http://example.org/data/Sotomayor"/>
  <exv:venue rdf:resource="http://example.org/data/PiscinaOlímpica"/>
  <rdf:type rdf:resource="http://example.org/vocab#OpenMarket"/>
</rdf:Description>
<rdf:Description rdf:about="http://example.org/data/Santiago">
```

RDF graph example in RDF/XML (cont.)

```
<exv:bus rdf:resource="http://example.org/data/ViñadelMar"/>
<exv:flight rdf:resource="http://example.org/data/Arica"/>
<exv:flight rdf:resource="http://example.org/data/ViñadelMar"/>
</rdf:Description>
<rdf:Description rdf:about="http://example.org/data/PiscinaOlímpica">
  <exv:city rdf:resource="http://example.org/data/Arica"/>
</rdf:Description>
<rdf:Description rdf:about="http://example.org/data/Sotomayor">
  <exv:city rdf:resource="http://example.org/data/ViñadelMar"/>
</rdf:Description>
<rdf:Description rdf:about="http://example.org/data/SantaLucía">
  <exv:city rdf:resource="http://example.org/data/Santiago"/>
</rdf:Description>
</rdf:RDF>
```

RDF graph example in JSON-LD

JSON-LD format:

- Abstract structure is similar to Turtle, but written using combination of lists and dictionaries.
- No IRI abbreviation.
- Nodes with IRI are indicated by "@id" key.
- Type relation is indicated by a special key: "@type".
- Literal values is indicated by a special key: "@value".

RDF graph example in JSON-LD (cont.)

```
[  
  {  
    "@id": "http://example.org/data/Santiago",  
    "http://example.org/vocab#bus": [  
      {  
        "@id": "http://example.org/data/ViñadelMar"  
      }  
    ],  
    "http://example.org/vocab#flight": [  
      {  
        "@id": "http://example.org/data/Arica"  
      },  
      {  
        "@id": "http://example.org/data/ViñadelMar"  
      }  
    ]  
  },  
]
```

RDF graph example in JSON-LD (cont.)

```
{  
  "@id": "http://example.org/data/SantaLucía",  
  "http://example.org/vocab#city": [  
    {  
      "@id": "http://example.org/data/Santiago"  
    }  
  ]  
,  
{  
  "@id": "http://example.org/data/EID15",  
  "@type": [  
    "http://example.org/vocab#OpenMarket",  
    "http://example.org/vocab#DrinksFestival",  
    "http://example.org/vocab#FoodFestival"  
  ],  
  "http://example.org/vocab#end": [  
    {  
      "  
    }  
  ]  
}
```

RDF graph example in JSON-LD (cont.)

```
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime",
    "@value": "2018-03-29T20:00:00"
  }
],
"http://example.org/vocab#name": [
  {
    "@value": "Nam"
  }
],
"http://example.org/vocab#start": [
  {
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime",
    "@value": "2018-03-22T12:00:00"
  }
],
"http://example.org/vocab#venue": [
  {
```

RDF graph example in JSON-LD (cont.)

```
        "@id": "http://example.org/data/SantaLucía"
    }
]
},
{
    "@id": "http://example.org/data/EID16",
    "@type": [
        "http://example.org/vocab#FoodFestival",
        "http://example.org/vocab#OpenMarket"
    ],
    "http://example.org/vocab#name": [
        {
            "@value": "Food Truck"
        }
    ],
    "http://example.org/vocab#venue": [
        {

```

RDF graph example in JSON-LD (cont.)

```
        "@id": "http://example.org/data/Sotomayor"
    },
    {
        "@id": "http://example.org/data/PiscinaOlímpica"
    }
]
},
{
    "@id": "http://example.org/data/PiscinaOlímpica",
    "http://example.org/vocab#city": [
        {
            "@id": "http://example.org/data/Arica"
        }
    ]
},
{
    "@id": "http://example.org/data/Sotomayor",
```

RDF graph example in JSON-LD (cont.)

```
"http://example.org/vocab#city": [
  {
    "@id": "http://example.org/data/ViñadelMar"
  }
],
{
  "@id": "http://example.org/data/Arica",
  "http://example.org/vocab#bus": [
    {
      "@id": "http://example.org/data/ViñadelMar"
    }
  ],
  "http://example.org/vocab#flight": [
    {
      "@id": "http://example.org/data/Santiago"
    }
]
```

RDF graph example in JSON-LD (cont.)

```
        ]
    },
{
  "@id": "http://example.org/data/ViñadelMar",
  "http://example.org/vocab#bus": [
    {
      "@id": "http://example.org/data/Santiago"
    },
    {
      "@id": "http://example.org/data/Arica"
    }
  ],
  "http://example.org/vocab#flight": [
    {
      "@id": "http://example.org/data/Santiago"
    }
  ]
}
```

RDF graph example in JSON-LD (cont.)

```
]} }
```

What about **blank node**?

(allowed as the subject and the object of an RDF triple)

Outline

1. Persistent Identifiers
2. Datatypes
3. Resource Description Framework (RDF)
4. Existential Nodes
5. Lexicalization

Blank nodes = anonymous nodes

- When modeling incomplete information or when “reifying” a relation into a node, we sometimes need to state that there must exist a node in the graph without being able or required to give an explicit identity to the node.

Blank nodes = anonymous nodes

- When modeling incomplete information or when “reifying” a relation into a node, we sometimes need to state that there must exist a node in the graph without being able or required to give an explicit identity to the node.
- In such a case, it may be useful to use the so-called **blank nodes**, which is a node in the graph to which an IRI is not assigned.

Blank nodes: Example

- Consider two co-located events chile:EID42 and chile:EID43 whose venue has yet to be announced.

Blank nodes: Example

- Consider two co-located events chile:EID42 and chile:EID43 whose venue has yet to be announced.
- Option 1: drop the ‘venue’ edge, but

Blank nodes: Example

- Consider two co-located events chile:EID42 and chile:EID43 whose venue has yet to be announced.
- Option 1: drop the ‘venue’ edge, but we can lose the information that these events have a venue and that both events have the same venue.

Blank nodes: Example

- Consider two co-located events chile:EID42 and chile:EID43 whose venue has yet to be announced.
- Option 1: drop the ‘venue’ edge, but we can lose the information that these events have a venue and that both events have the same venue.
- Option 2: create a fresh IRI representing the venue, but

Blank nodes: Example

- Consider two co-located events chile:EID42 and chile:EID43 whose venue has yet to be announced.
- Option 1: drop the ‘venue’ edge, but we can lose the information that these events have a venue and that both events have the same venue.
- Option 2: create a fresh IRI representing the venue, but this becomes semantically indistinguishable from there being a known venue (though right now it should be unknown).

Blank nodes: Example

- Consider two co-located events chile:EID42 and chile:EID43 whose venue has yet to be announced.
- Option 1: drop the ‘venue’ edge, but we can lose the information that these events have a venue and that both events have the same venue.
- Option 2: create a fresh IRI representing the venue, but this becomes semantically indistinguishable from there being a known venue (though right now it should be unknown).
- Option 3: use a blank node

Blank nodes: Example (contd.)

Using blank nodes, we can model the two co-located events as:



- Edges capture the meaning that there exists a common venue for chile:EID42 and chile:EID43 without identifying it.
- Can be written in Turtle (assuming namespace prefixes defined) as:

```
chile:EID42 chile:venue _:b1 .  
chile:EID43 chile:venue _:b1 .
```

where _:b1 is the blank node identifier.

Identity of blank nodes

- Does blank nodes have an identifier?

Identity of blank nodes

- Does blank nodes have an identifier? Yes. (See the N-triple version of the previous example).

Identity of blank nodes

- Does blank nodes have an identifier? Yes. (See the N-triple version of the previous example).
- How does the identifier differ from the standard IRIs?

Identity of blank nodes

- Does blank nodes have an identifier? Yes. (See the N-triple version of the previous example).
- How does the identifier differ from the standard IRIs?
 - They differ in scope: blank node identifiers are unique within a single RDF document.

Identity of blank nodes

- Does blank nodes have an identifier? Yes. (See the N-triple version of the previous example).
- How does the identifier differ from the standard IRIs?
 - They differ in scope: blank node identifiers are unique within a single RDF document.
 - Blank node identifiers cannot be referred to from outside the document
 - Re-writing/reloading the graph (by machines) may change blank node identifiers.

Many-value relation representation with blank nodes

Model the following in RDF: “Chutney has 1 lb. green Mango and 1 tsp. Cayenne pepper as ingredients.”

Many-value relation representation with blank nodes

Attempt 1:

```
@prefix ex: <http://example.org/> .  
  
ex:Chutney ex:hasIngredient "1 lb. green mango",  
           "1 tsp. Cayenne pepper"
```

Many-value relation representation with blank nodes

Attempt 1:

```
@prefix ex: <http://example.org/> .
```

```
ex:Chutney ex:hasIngredient "1 lb. green mango",  
      "1 tsp. Cayenne pepper"
```

Can we query all recipes containing green mango?

Many-value relation representation with blank nodes

Attempt 2:

```
@prefix ex: <http://example.org/> .  
ex:Chutney ex:ingredient ex:greenMango;  
          ex:amount "1 lb." ;  
          ex:ingredient ex:CayennePepper;  
          ex:amount "1tsp." .
```

Many-value relation representation with blank nodes

Attempt 2:

```
@prefix ex: <http://example.org/> .  
ex:Chutney ex:ingredient ex:greenMango;  
          ex:amount "1 lb." ;  
          ex:ingredient ex:CayennePepper;  
          ex:amount "1tsp." .
```

Can we unambiguously obtain the amount of Cayenne pepper used by Chutney?

Many-value relation representation with blank nodes

Attempt 3:

```
@prefix ex: <http://example.org/> .  
  
ex:Chutney ex:ingredient ex:greenMango,  
            ex:CayennePepper .  
ex:greenMango ex:amount "1 lb." ;  
ex:CayennePepper ex:amount "1 tsp."
```

Many-value relation representation with blank nodes

Attempt 3:

```
@prefix ex: <http://example.org/> .  
  
ex:Chutney ex:ingredient ex:greenMango,  
            ex:CayennePepper .  
ex:greenMango ex:amount "1 lb." ;  
ex:CayennePepper ex:amount "1 tsp."
```

Can we unambiguously obtain the amount of Cayenne pepper used by Chutney?

Many-value relation representation with blank nodes

Attempt 4 without blank nodes:

```
@prefix ex: <http://example.org/> .  
  
ex:Chutney ex:hasIngredient ex:ingredient1, ex:ingredient2 .  
ex:ingredient1 ex:ingredient ex:greenMango;  
              ex:amount "1 lb." .  
ex:ingredient2 ex:ingredient ex:CayennePaper ;  
              ex:amount "1 tsp." .
```

Many-value relation representation with blank nodes

Attempt 4 with blank nodes:

```
@prefix ex: <http://example.org/> .  
  
ex:Chutney ex:hasIngredient _:b1, _:b2 .  
_:b1 ex:ingredient ex:greenMango;  
      ex:amount "1 lb." .  
_:b2 ex:ingredient ex:CayennePaper ;  
      ex:amount "1 tsp." .
```

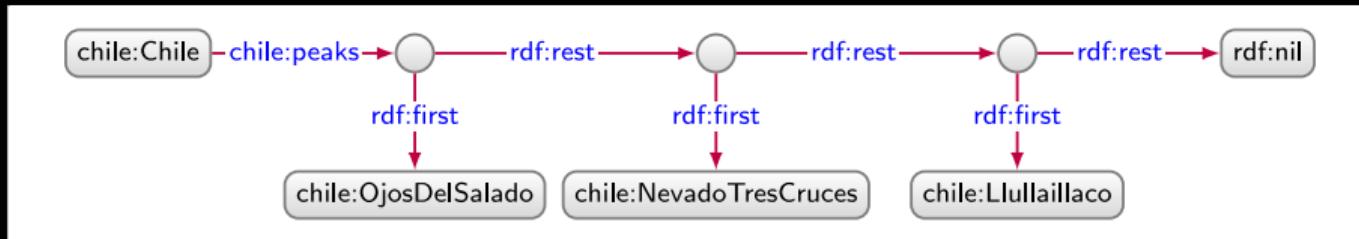
Many-value relation representation with blank nodes

Attempt 4 with blank nodes using square bracket syntax:

```
@prefix ex: <http://example.org/> .
```

```
ex:Chutney ex:hasIngredient
  [ ex:ingredient ex:greenMango ;
    ex:amount "1 lb." ],
  [ ex:ingredient ex:CayennePaper ;
    ex:amount "1 tsp." ].
```

RDF list representation with blank nodes



can be compactly represented in Turtle using parentheses notation:

`chile:Chile chile:peaks`

`(chile:OjosDelSalado chile:NevadoTresCruces chile:Llullaillaco) .`

Further remarks on blank nodes

- Existential nodes/blank nodes can be convenient, but also complicate operations on graphs.
- In some cases, we may want to replace blank nodes with nodes with canonical IRIs.
- Some researchers also suggest to minimize the use of blank nodes in the graph.

Outline

1. Persistent Identifiers
2. Datatypes
3. Resource Description Framework (RDF)
4. Existential Nodes
5. Lexicalization

Lexicalization

- Global identifiers (IRIs) may sometimes have a human-interpretable form, e.g., chile:Santiago.
- But, the identifier strings themselves do not carry any formal semantic significance
 - It's perfectly acceptable to simply use random string as identifier as long as its use is unambiguous.
- Real world examples: in Wikidata, the identifier for Eswatini is wd:Q1050
 - No need to choose between languages for creating IRIS, e.g., wd:Eswatini (English), wd:eSwatini (Swahili), or wd:Esuatini (Spanish).

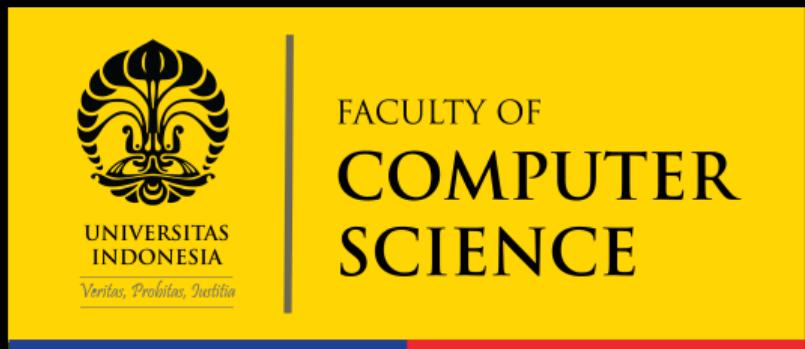
Lexicalization (cont.)

- Since identifiers can be arbitrary, it is common to add edges that provide human-interpretable label for nodes (possibly with language tags), e.g.,

```
wd:Q1050 rdfs:label "Swatini"  
wd:Q1050 rdfs:label "Swatini"@en  
wd:Q1050 rdfs:label "eSwatini"@sw  
wd:Q1050 rdfs:label "Esuaatini"@es
```

Lexicalization (cont.)

- Other linguistic information could also be added, such as aliases (using `skos:altLabel` property) or comments (using `rdfs:comment` property)
- Benefits of linguistic information through such metadata:
 - can help user identify which real-world entity a node in KG actually references;
 - enables cross-referencing with text corpora to find documents that provide details about an entity;
 - can help user interfaces in displaying the data.



Semantic Web 04: Querying Data Graph

Adila Krisnadhi – adila@cs.ui.ac.id Faculty of Computer Science, Universitas
Indonesia

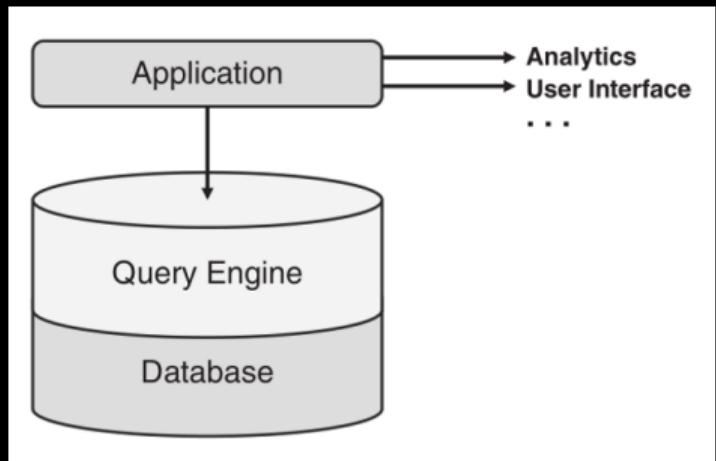
Outline

1. Application Architecture
2. Basic Graph Patterns
3. Complex Graph Patterns
4. Navigational Graph Patterns: Property path
5. SPARQL Output Forms

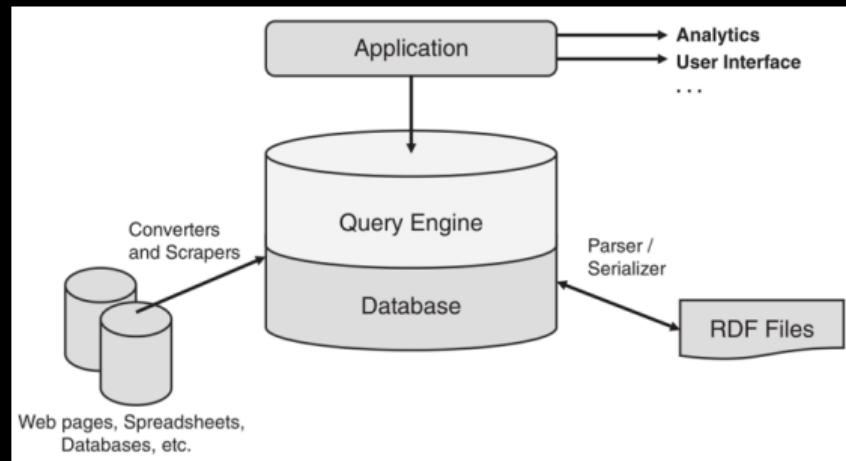
- Where do RDF data come from?
 - Manually created.
 - RDF files provided by others ⇒ we need **RDF parsers** and **RDF serializers** for reading and writing RDF files.
 - Non-RDF files (spreadsheets, web pages, etc.) ⇒ we need **converters** and **scrapers**.
 - Relational databases ⇒ we need either a **wrapper** over the database or a way to convert data from relational data model to RDF.

- How do we retrieve information/data from RDF graph?
 - Searching and text processing from files (may be complicated).
 - Better: **querying** if RDF is stored in a database (for RDF, not relational) ⇒ need **RDF stores and query engines**.
- How do we utilize richer semantic represented (formally) in RDF data, if any?
 - Use **reasoning engines**

RDF application vs. traditional database application



Traditional database application
(database = RDBMS)



RDF application
(database = RDF graph store or RDBMS with
RDF wrapper)

RDF parsers and serializers

- **RDF parsers**: read triples from an RDF file into an internal representation of the triple.
- **RDF serializers**: write RDF triples from their internal representation into memory or files.
- The same library may contain both parsing and serialization functions.
- The same RDF graph (set of triples) can be represented by many possible files.
 - Different serialization formats (n-triples, turtle, json-ld, etc.) Different ordering of triples written in the file.
 - Different blank node IDs used in different files But they're ALL the same

RDF software libraries

- Apache Jena <http://jena.apache.org/>
- Apache any23 <http://any23.apache.org/>
- rdflib <https://rdflib.readthedocs.io/en/stable/>
- Eclipse rdf4j <https://rdf4j.org/>
- Redland <https://librdf.org/>
- JsonLD <https://github.com/lanthaler/JsonLD>
- NxParser <https://github.com/nxparser/nxparser>
- RDFSharp <https://github.com/mdesalvo/RDFSharp>
- See <https://www.w3.org/2001/sw/wiki/Tools> for a (not always up-to-date) list.

RDF stores a.k.a triple stores

- RDF store stores RDF data (analogous to RDBMS for relational databases)
 - Typically also includes parser and serializer.
 - Unlike RDBMS, RDF store provides key ability to merge two RDF datasets together.
- Some RDF stores are extensions of traditional RDBMS, storing RDF triples in tables as:
 - a single relation of arity three, i.e., a **triple table**; or
 - a collection of binary relations/tables, one for each property, i.e., **vertical partitioning**; or
 - a collection of n -ary relations/tables, each contains entities of the same type, i.e., **property tables**.

RDF stores a.k.a triple stores (cont.)

- RDF standardization is established earlier than many RDF stores. Hence, the underlying data model is shared by ALL of RDF store products.
 - Easy to move/migrate data from one RDF store to another.
 - Simplifies effort of data federation between multiple RDF stores.

RDF query engines

- Data in RDF store can be accessed using SPARQL query.
- Query is processed by RDF query engine.
 - Every RDF store comes with its own query engine.
 - Query is run on the SPARQL engine endpoints, not on plain RDF files.
- SPARQL includes protocol to communicate queries and query results.
 - Implemented by query engine via SPARQL endpoints.
 - RDF data can thus also come from such endpoints.
- Some query engines allow translation of SPARQL queries into SQL, hence allowing access to databases that are not triple stores.

Reasoning engines a.k.a. reasoners

- Provides capability to infer logical consequences from RDF data and schemata.
- Often closely related or even integrated with RDF query engine.
- Reasoning is based on particularly chosen standardized semantics, e.g., RDF Schema, SHACL, variants of OWL.

Data federation

- RDF data model is designed from the beginning with data federation in mind.
 - Converting information (from any source) to RDF triples enables data federation.
- Strategy 1 (as seen in RDF application architecture):
 - convert information from multiple sources into single format
 - combine those information in a single triple store to be queried from application.
- Strategy 2: application queries multiple triple stores separately.
 - Possible because all triple stores share the same data model.
 - Queries in application need not know where a particular triple came from – as we assume all data are in a federated graph.

Examples of RDF application

- RDF-backed web portals – construct web pages from RDF data, possibly from multiple RDF stores.
- Calendar integration – shows appointments from different people and teams on a single calendar view.
- Map integration – shows locations of points of interest gathered from different web sites, spreadsheets, and databases all on a single map.
- Annotation – allows a community of users to apply keywords (with URIs) to information (tagging) for others to consult.
- Content management – makes a single index of information resources (documents, web pages, databases, etc.) that are available in several content stores.

Our focus:
Querying

Outline

1. Application Architecture
2. Basic Graph Patterns
3. Complex Graph Patterns
4. Navigational Graph Patterns: Property path
5. SPARQL Output Forms

Basic graph patterns

- Core of structured graph query languages: basic graph pattern (BGP).
 - Follow the same model as the data graph being queried, but **additionally allowing variables** as terms.
 - Nodes **and** edges may be replaced with variables.
 - May form cycles.

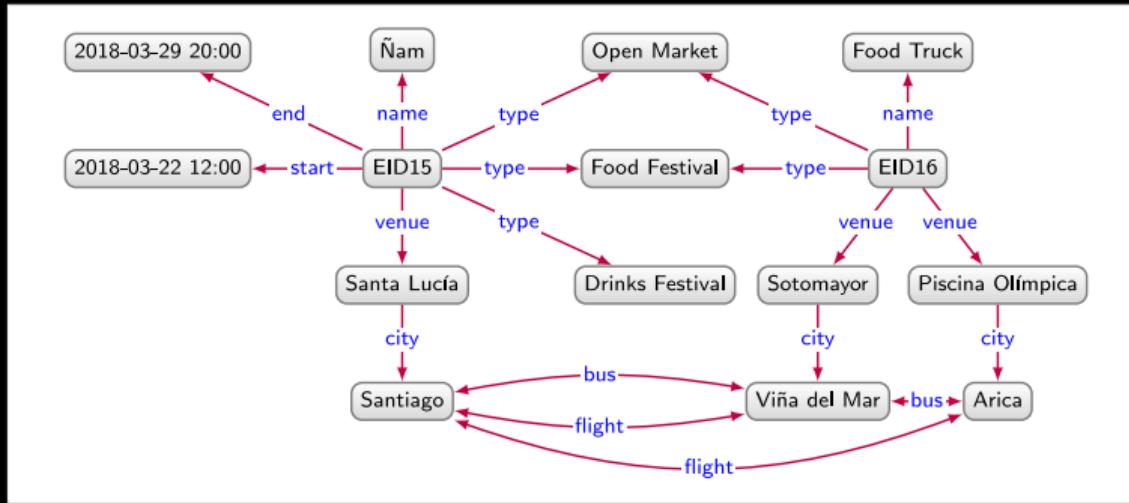
Basic graph patterns

- Core of structured graph query languages: **basic graph pattern (BGP)**.
 - Follow the same model as the data graph being queried, but **additionally allowing variables** as terms.
 - Nodes **and** edges may be replaced with variables.
 - May form cycles.
- For RDF/DELG, a basic graph pattern corresponds to set of **triple patterns**, i.e., a set of triples that allow variables (indicated with question mark '?') in the subject, predicate, and object positions.

BGP evaluation

- **Evaluation of BGP:** generate mappings from the BGP to constants (nodes or edges) in the data graph such that the image of the BGP under the mapping (by replacing variables with constants) is contained in the data graph.
- Semantics of the evaluation for RDF BGP, i.e., in SPARQL, is **homomorphism-based** where multiple variables can be mapped to the same terms
 - Note: some other graph query languages, e.g., Cypher for property graphs, employ **isomorphism-based semantics** where variables must be mapped to unique terms.

Example



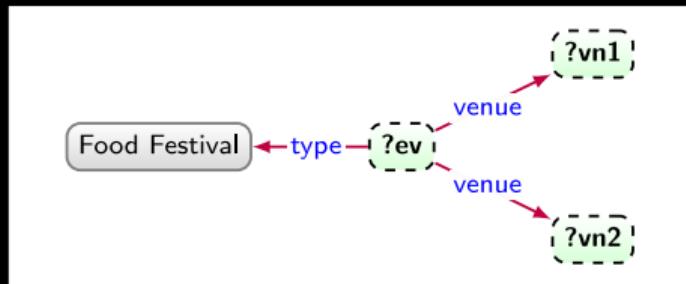
We wish to query food festivals. The answer should include two, possibly the same, venues of the events.

Example (cont.)

Then the corresponding BGP is:

Example (cont.)

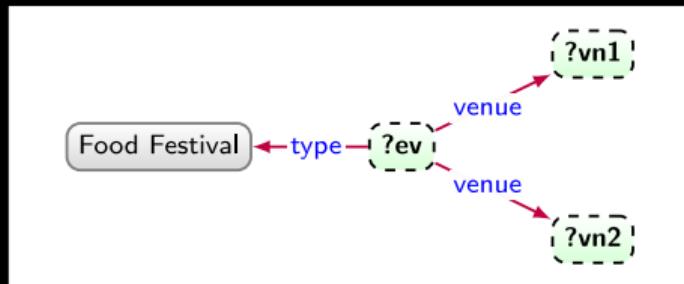
Then the corresponding BGP is:



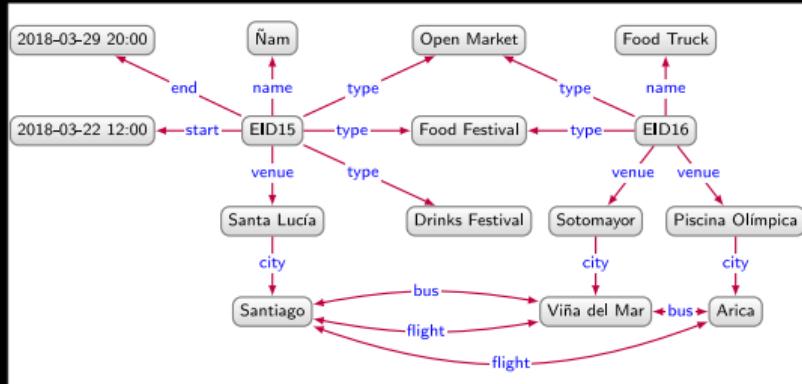
Example (cont.)

Then the corresponding BGP is:

Yielding answers ...



Example (cont.)

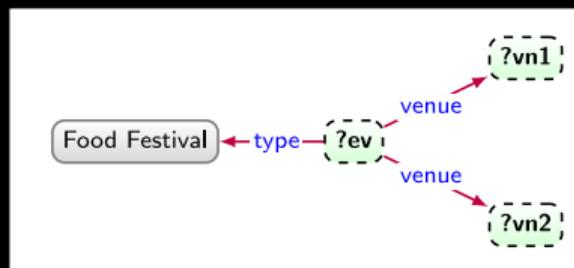


Yielding answers (with homomorphism-based semantics):

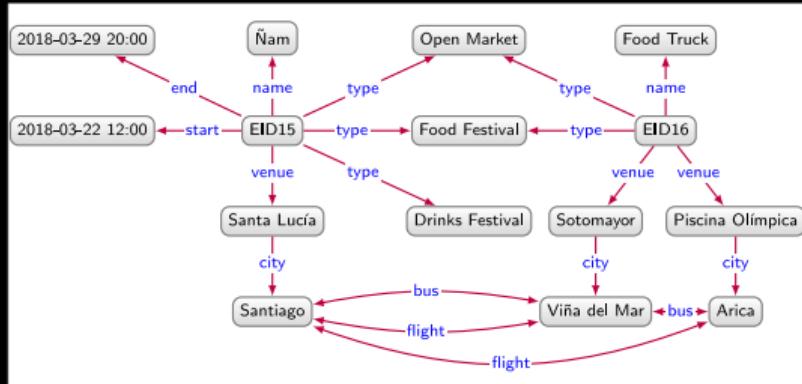
?ev

?vn1

?vn2

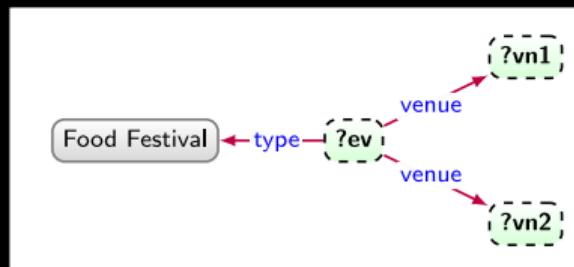


Example (cont.)

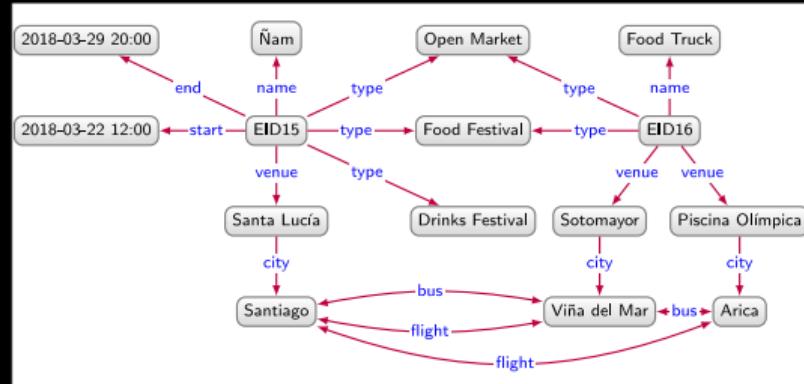


Yielding answers (with homomorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor

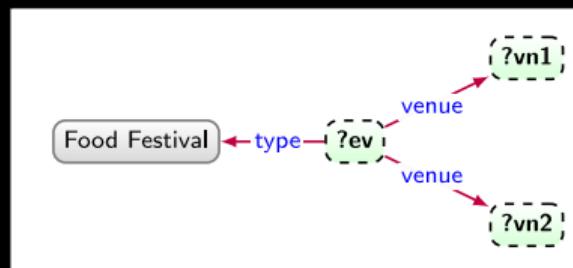


Example (cont.)

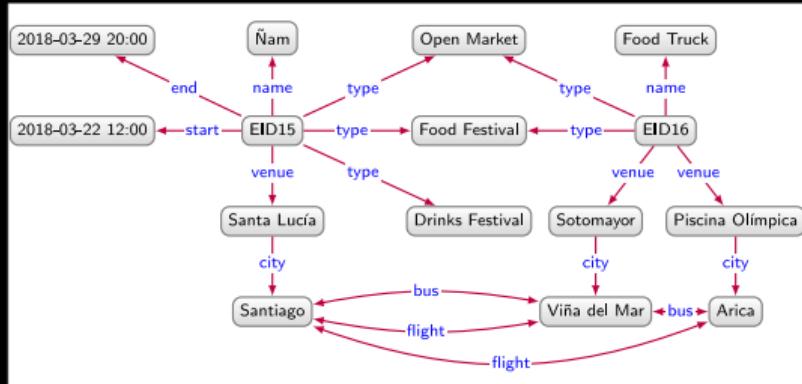


Yielding answers (with homomorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica

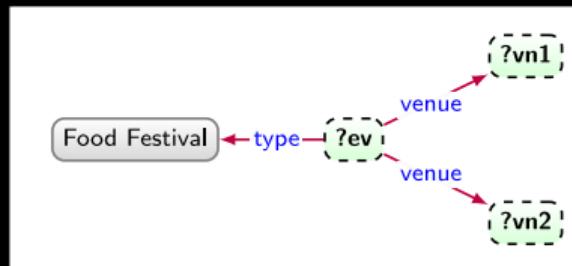


Example (cont.)

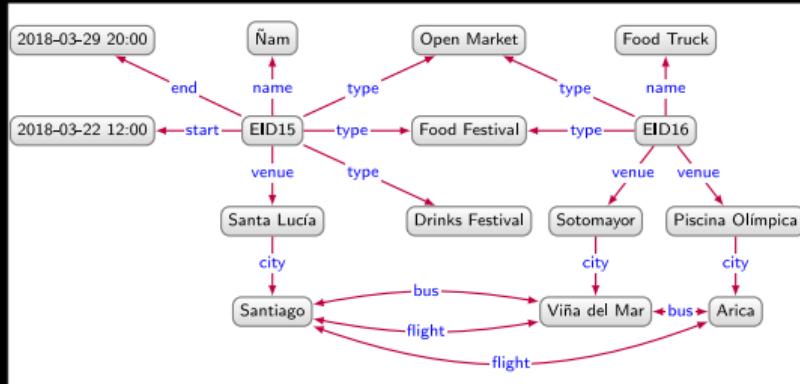


Yielding answers (with homomorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica

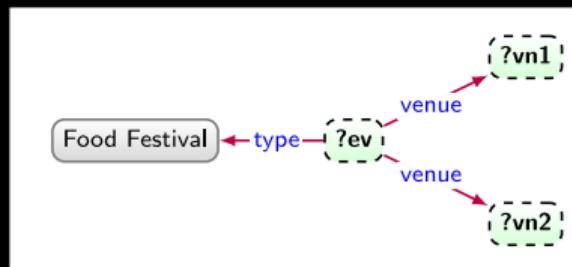


Example (cont.)

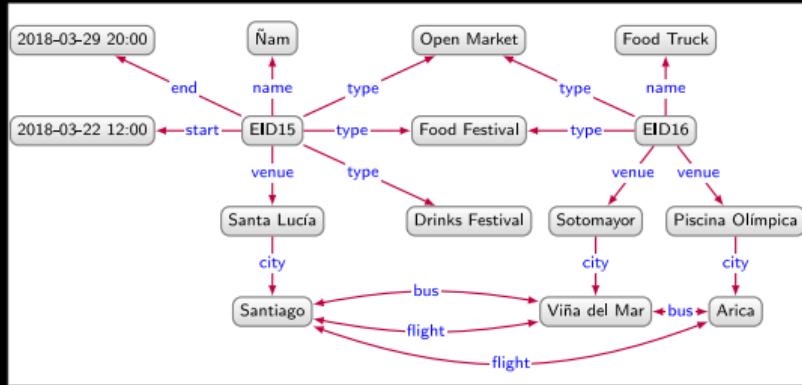


Yielding answers (with homomorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica
EID16	Sotomayor	Sotomayor

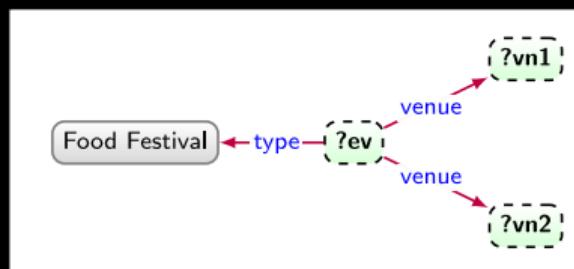


Example (cont.)

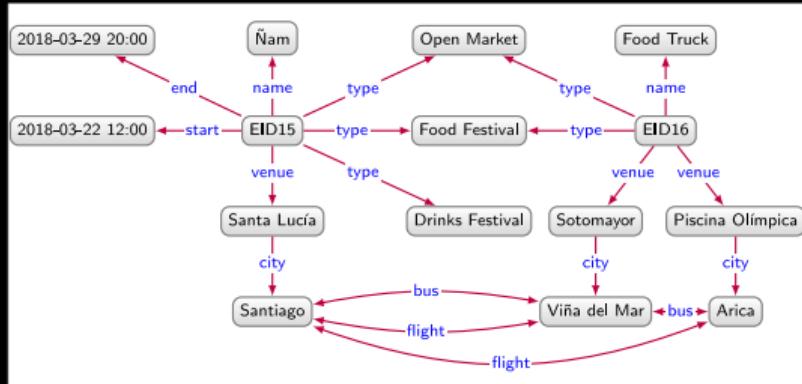


Yielding answers (with homomorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica
EID16	Sotomayor	Sotomayor
EID15	Santa Lucía	Santa Lucía



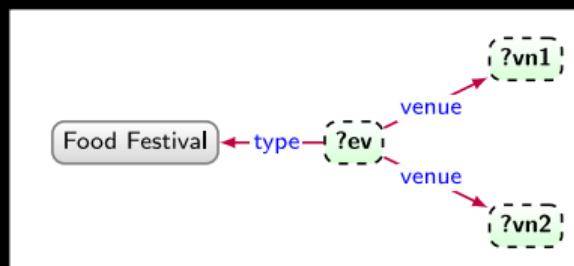
Example (cont.)



Yielding answers (with homomorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica
EID16	Sotomayor	Sotomayor
EID15	Santa Lucía	Santa Lucía

In isomorphism-based semantics, the last three mappings are not included as answers.



In SPARQL ...

The data:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix ex: <http://example.org/data/> .  
@prefix exv: <http://example.org/vocab#> .  
  
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival, ex:DrinksFestival ;  
    exv:name "Ñam" ;  
    exv:start "2018-03-22T12:00:00"^^xsd:dateTime ;  
    exv:end "2018-03-29T20:00:00"^^xsd:dateTime ;  
    exv:venue ex:SantaLucía .  
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;  
    exv:name "Food Truck"^^xsd:string ;  
    exv:venue ex:Sotomayor, ex:PiscinaOlímpica .  
  
ex:SantaLucía exv:city ex:Santiago .  
ex:Sotomayor exv:city ex:ViñadelMar .  
...
```

In SPARQL ... (cont.)

The query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/
PREFIX ex: <http://example.org/data/>
PREFIX exv: <http://example.org/vocab#>
```

```
select ?ev ?vn1 ?vn2
where {
    ?ev rdf:type exv:FoodFestival ;
          exv:venue ?vn1, ?vn2 .
}
```

- Mapping 1:
?ev = <http://example.org/data/EID16>
?vn1 = <http://example.org/data/PiscinaOlímpica>
?vn2 = <http://example.org/data/Sotomayor>
- Mapping 2:
?ev = <http://example.org/data/EID16>
?vn1 = <http://example.org/data/Sotomayor>
?vn2 = <http://example.org/data/PiscinaOlímpica>
- Mapping 3: ...
- ...

About SPARQL

- SPARQL = SPARQL Protocol and RDF Query Language.
- Query language for RDF graphs.
- SPARQL 1.0: W3C Specification in 2008
- SPARQL 1.1: W3C Specification in 2013

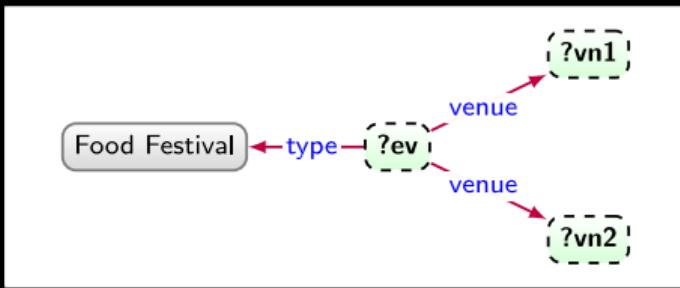
About SPARQL 1.1

- Standard: <https://www.w3.org/TR/sparql11-overview/>
- Query language (syntax and semantics)
- RDF graph update through SPARQL
- Graph Store HTTP Protocol: HTTP operations for managing a graph collection
- Entailment regimes: query results with (additional) inferences
- Service description: methods for discovering and describing (using standard vocabulary) SPARQL services
- Query federation: querying over distributed sources (multiple endpoints)
- Query result format standards in XML, JSON, CSV, TSV.

Basic graph patterns

- VAR: set of variables
- IRI: set of IRIs
- CON: set of constants $\rightarrow \text{CON} = \text{IRI} \cup \text{BN} \cup \text{LIT}$
- VAR, IRI, BN, and LIT are pairwise disjoint.
- A triple pattern is a triple (s, p, o) where
 - subject $s \in \text{VAR} \cup \text{IRI} \cup \text{BN}$ – (variables, IRI, or blank nodes)
 - predicate $p \in \text{VAR} \cup \text{IRI}$ – (variables or IRIs)
 - object $o \in \text{VAR} \cup \text{IRI} \cup \text{BN} \cup \text{LIT}$ – (variables, IRI, blank nodes, or literals)
- A basic graph pattern (BGP) is a set of triple patterns.

Example



The BGP consists of 3 triple patterns:

$\{(\text{?ev, type, Food Festival}),$
 $(\text{?ev, venue, ?vn1}),$
 $(\text{?ev, venue, ?vn2})\}$

BGP evaluation

- Answering query = evaluating BGP over the data graph. How?
 - Find all possible ways to instantiate variables in the BGP such that each of those instantiations results in a subgraph of the RDF data graph.
 - Instantiation \rightsquigarrow mapping from variables to constants.
 - Correct instantiation \rightsquigarrow solution mapping.
 - Set of correct instantiations \rightsquigarrow query answer.

BGP evaluation (cont.)

- For a BGP Q , $\text{VAR}(Q)$ denotes the set of all variables appearing in Q .
- For a partial mapping $\mu : \text{VAR} \rightarrow \text{CON}$ (from variables to constants)
 - $\text{dom}(\mu)$ denotes the domain of μ , i.e., the set of variables for which μ is defined
 - given a variable $v \in \text{dom}(\mu)$, $\mu(v)$ is a constant (IRI, blank node, or literal),
 - given a BGP Q , $\mu(Q)$ is the set of triple patterns in which any occurrences of variable $v \in \text{VAR}(Q) \cap \text{dom}(\mu)$ is replaced in Q by the constant $\mu(v)$.
- $\text{VAR}(Q)$ may contain variables NOT in $\text{dom}(\mu)$ and vice versa.
 - If $\text{VAR}(Q) \subseteq \text{dom}(\mu)$ holds, $\mu(Q)$ is in fact an RDF data graph (because it does not contain any variable).
 - If $\text{dom}(\mu) \subsetneq \text{VAR}(Q)$, $\mu(Q)$ remains a BGP (because it still contains variables).

BGP evaluation (cont.)

- A partial mapping $\mu : \text{VAR} \rightarrow \text{CON}$ is called a **solution mapping** of a BGP Q over an RDF data graph G if and only if $\mu(Q)$ is a **subgraph** of G .
 - Note: if G_1 and G_2 are two RDF graphs (sets of RDF triples), then we say that G_1 is a subgraph of G_2 if and only if $G_1 \subseteq G_2$.
- Given a BGP/query Q and RDF data graph G , the **answer** for Q with respect to G , denoted by $Q(G)$, is a **multiset of solution mappings** defined as:

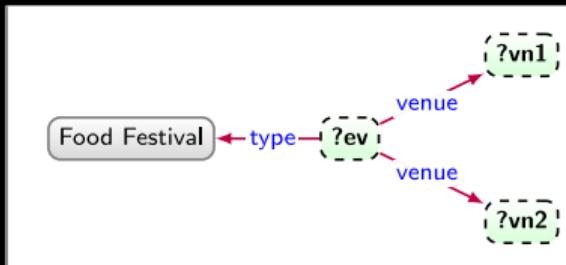
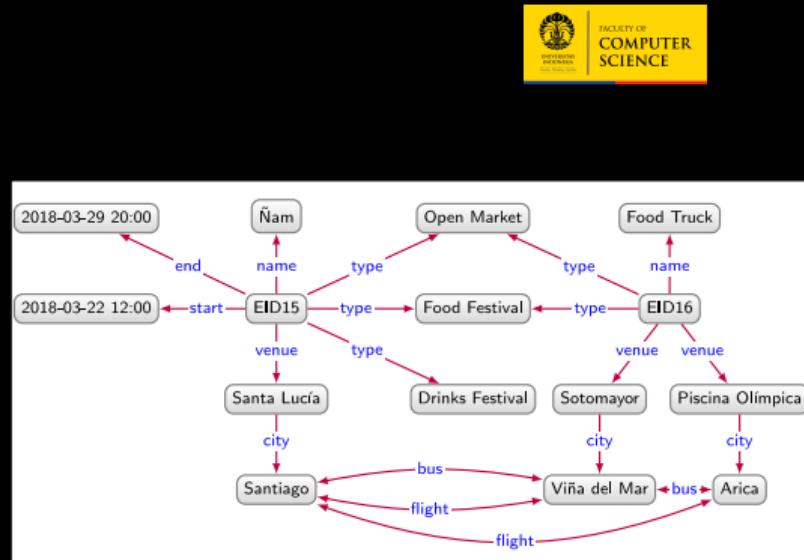
$$Q(G) = \{\mu \mid \mu(Q) \subseteq G \text{ and } \text{dom}(\mu) \subseteq \text{VAR}(Q)\}$$

It's a multiset: duplicates of solution mappings are allowed and solution mappings are unordered.

Example

Why is this the correct answer for the BGP w.r.t. the data graph on the right?

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica
EID16	Sotomayor	Sotomayor
EID15	Santa Lucía	Santa Lucía



Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev}, \text{type Food Festival}), (\text{?ev}, \text{venue}, \text{?vn1}), (\text{?ev}, \text{venue}, \text{?vn2})\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$?

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Priscina Ol\'impica})\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$?

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$? Yes.

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$? Yes.
3. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$? Yes.
3. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica})\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev}, \text{type Food Festival}), (\text{?ev}, \text{venue, ?vn1}), (\text{?ev}, \text{venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, \text{type, Food Festival}), (EID16, \text{venue, Priscina Ol\'impica}), (EID16, \text{venue, Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, \text{type, Food Festival}), (EID16, \text{venue, Sotomayor}), (EID16, \text{venue, Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$? Yes.
3. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, \text{type, Food Festival}), (EID16, \text{venue, Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$?

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

1. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica}), (EID16, venue, \text{Sotomayor})\}$ Is $\mu(Q) \subseteq G$? Yes.
2. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$? Yes.
3. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Piscina Ol\'impica}, \text{?vn2} \mapsto \text{Piscina Ol\'impica}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Priscina Ol\'impica})\}$
Is $\mu(Q) \subseteq G$? Yes.

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$

$$\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$

$$\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$$

Is $\mu(Q) \subseteq G$?

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$

$$\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$$

Is $\mu(Q) \subseteq G$? Yes.

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$
Is $\mu(Q) \subseteq G$? Yes.
5. $\mu = \{\text{?ev} \mapsto EID15, \text{?vn1} \mapsto \text{Santa Luc\'ia}, \text{?vn2} \mapsto \text{Santa Luc\'ia}\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$
Is $\mu(Q) \subseteq G$? Yes.
5. $\mu = \{\text{?ev} \mapsto EID15, \text{?vn1} \mapsto \text{Santa Luc\'ia}, \text{?vn2} \mapsto \text{Santa Luc\'ia}\}$
 $\mu(Q) = \{(EID15, type, \text{Food Festival}), (EID15, venue, \text{Santa Luc\'ia})\}$

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$
Is $\mu(Q) \subseteq G$? Yes.
5. $\mu = \{\text{?ev} \mapsto EID15, \text{?vn1} \mapsto \text{Santa Luc\'ia}, \text{?vn2} \mapsto \text{Santa Luc\'ia}\}$
 $\mu(Q) = \{(EID15, type, \text{Food Festival}), (EID15, venue, \text{Santa Luc\'ia})\}$
Is $\mu(Q) \subseteq G$?

Example (cont.)

Data graph $G = \{(EID, name, \tilde{N}am), \dots, (EID15, type, \text{Food Festival}), (EID, venue, \text{Santa Luc\'ia}), \dots, (EID16, type, \text{Food Festival}), \dots, (EID16, venue, \text{Sotomayor}), (EID16, venue, \text{Piscina Ol\'impica}), \dots\}$

BGP $Q = \{(\text{?ev, type Food Festival}), (\text{?ev, venue, ?vn1}), (\text{?ev, venue, ?vn2})\}$

4. $\mu = \{\text{?ev} \mapsto EID16, \text{?vn1} \mapsto \text{Sotomayor}, \text{?vn2} \mapsto \text{Sotomayor}\}$
 $\mu(Q) = \{(EID16, type, \text{Food Festival}), (EID16, venue, \text{Sotomayor})\}$
Is $\mu(Q) \subseteq G$? Yes.
5. $\mu = \{\text{?ev} \mapsto EID15, \text{?vn1} \mapsto \text{Santa Luc\'ia}, \text{?vn2} \mapsto \text{Santa Luc\'ia}\}$
 $\mu(Q) = \{(EID15, type, \text{Food Festival}), (EID15, venue, \text{Santa Luc\'ia})\}$
Is $\mu(Q) \subseteq G$? Yes.

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?
Yes.

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?
Yes.
 - Happens when we cannot find any solution mapping μ such that $\mu(Q) \subseteq G$.

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?
Yes.
 - Happens when we cannot find any solution mapping μ such that $\mu(Q) \subseteq G$.
- Can we have an answer that contains an empty solution mapping?

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?
Yes.
 - Happens when we cannot find any solution mapping μ such that $\mu(Q) \subseteq G$.
- Can we have an answer that contains an empty solution mapping?
Yes.

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?
Yes.
 - Happens when we cannot find any solution mapping μ such that $\mu(Q) \subseteq G$.
- Can we have an answer that contains an empty solution mapping?
Yes. Happens when:
 - $\text{VAR}(Q) = \emptyset$, i.e., the query contains no variable;
 - the projection variables of the query is disjoint with the $\text{VAR}(Q)$ (see discussion on projection later)

Empty answer vs. empty mappings

- Can an answer to a BGP be empty (i.e., the query has no answer)?
Yes.
 - Happens when we cannot find any solution mapping μ such that $\mu(Q) \subseteq G$.
- Can we have an answer that contains an empty solution mapping?
Yes. Happens when:
 - $\text{VAR}(Q) = \emptyset$, i.e., the query contains no variable;
 - the projection variables of the query is disjoint with the $\text{VAR}(Q)$ (see discussion on projection later)
- Watch out: empty answer IS NOT EQUAL TO answer with an empty solution mapping.

Example (assume namespace prefix is already defined)

```
ex:EID14 rdf:type exv:MusicFestival .  
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival, exv:DrinksFestival .  
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival .
```

```
SELECT * WHERE {  
  ?event rdf:type exv:MusicFestival,  
          exv:ClosedMarket ;  
  exv:name ?name ;  
  exv:venue ?ven .  
}
```

returns an empty answer, i.e., the query has no answer.

Example (assume namespace prefix is already defined)

```
ex:EID14 rdf:type exv:MusicFestival .  
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival, exv:DrinksFestival .  
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival .
```

```
SELECT * WHERE {  
    ex:EID14 rdf:type exv:MusicFestival .  
}
```

returns an answer containing exactly one empty solution mapping because the BGP has no variable.

Blank nodes in query

Blank nodes are allowed in the graph pattern.

- May appear in the subject or object position of a triple pattern.
- Are given arbitrary IDs.
- Act like variables, but cannot be projected by the SELECT clause.

Blank nodes in query

Blank nodes are allowed in the graph pattern.

- May appear in the subject or object position of a triple pattern.
- Are given arbitrary IDs.
- Act like variables, but cannot be projected by the SELECT clause.

Blank nodes may appear in query answer/solution mappings.

- Represent some unknown entities (that exist).
- Are given arbitrary IDs that may be different from its ID in the input RDF graph; repeated occurrences in the answer denote the same entities.

Example

The following two queries are equivalent.

```
SELECT ?event ?city WHERE {  
    ?event rdf:type ?type ;  
        exv:venue ?ven .  
    ?ven exv:city ?city .  
}
```

```
SELECT ?event ?city WHERE {  
    ?event rdf:type [] ;  
        exv:venue  
            [ exv:city ?city ] .  
}
```

BGP evaluation with blank nodes

- With blank nodes in the BGP, BGP evaluation needs to be modified.
- Intuition: solution mappings need to account for blank nodes in the BGP that must be mapped to some constants in the data graph.

BGP evaluation with blank nodes (cont.)

- Let Q be a BGP and G an RDF graph where Q may contain blank nodes.
- A partial mapping $\mu: \text{VAR} \cup \text{BN} \rightarrow \text{CON}$ is a **solution mapping** for Q with respect to G iff $\mu(Q) \subseteq G$.
 - We only extend the domain $\text{dom}(\mu)$ of μ to also allow blank nodes.
 - We define $\text{domvar}(\mu) = \text{dom}(\mu) \cap \text{VAR}$ – the set of variables for which μ is defined.
 - We define $\mu_{\text{var}} = \{v \mapsto \mu(v) \mid v \in \text{VAR}\}$ – the mapping μ restricted only to variables of μ (removing all the mapping for blank nodes).
- The definition for query answer becomes as follows: the **answer** for Q with respect to G , denoted $Q(G)$ is a **multiset of solution mappings** defined as:

$$Q(G) = \{\mu_{\text{var}} \mid \mu(Q) \subseteq G \text{ and } \text{domvar}(\mu) \subseteq \text{VAR}(Q)\}$$

Queries involving datatypes

To match literals, the datatype must also match.

```
ex:ex1 exv:p "test" .  
ex:ex2 exv:p "test"^^xsd:string .  
ex:ex3 exv:p "test"@en .  
ex:ex4 exv:p "42"^^xsd:integer .  
ex:ex5 exv:p 42 .
```

```
SELECT * WHERE {  
    ?s exv:p "test" .  
}
```

Answer:

Queries involving datatypes

To match literals, the datatype must also match.

```
ex:ex1 exv:p "test" .  
ex:ex2 exv:p "test"^^xsd:string .  
ex:ex3 exv:p "test"@en .  
ex:ex4 exv:p "42"^^xsd:integer .  
ex:ex5 exv:p 42 .
```

```
SELECT * WHERE {  
    ?s exv:p "test" .  
}
```

Answer:

```
{ {?s ↦ ex:ex1}, {?s ↦ ex:ex2} }
```

Queries involving datatypes

To match literals, the datatype must also match. The datatype of language-tagged strings is `rdf:langString`, not `xsd:string`.

```
ex:ex1 exv:p "test" .  
ex:ex2 exv:p "test"^^xsd:string .  
ex:ex3 exv:p "test"@en .  
ex:ex4 exv:p "42"^^xsd:integer .  
ex:ex5 exv:p 42 .
```

```
SELECT * WHERE {  
    ?s exv:p "test"@en .  
}
```

Answer:

Queries involving datatypes

To match literals, the datatype must also match. The datatype of language-tagged strings is `rdf:langString`, not `xsd:string`.

```
ex:ex1 exv:p "test" .  
ex:ex2 exv:p "test"^^xsd:string .  
ex:ex3 exv:p "test"@en .  
ex:ex4 exv:p "42"^^xsd:integer .  
ex:ex5 exv:p 42 .
```

```
SELECT * WHERE {  
    ?s exv:p "test"@en .  
}
```

Answer:
{ {?s ↦ ex:ex3} }

Queries involving datatypes

To match literals, the datatype must also match. For numeric values, some syntactic sugar is allowed.

```
ex:ex1 exv:p "test" .  
ex:ex2 exv:p "test"^^xsd:string .  
ex:ex3 exv:p "test"@en .  
ex:ex4 exv:p "42"^^xsd:integer .  
ex:ex5 exv:p 42 .
```

```
SELECT * WHERE {  
    ?s exv:p 42 .  
}
```

Answer:

Queries involving datatypes

To match literals, the datatype must also match. For numeric values, some syntactic sugar is allowed.

```
ex:ex1 exv:p "test" .  
ex:ex2 exv:p "test"^^xsd:string .  
ex:ex3 exv:p "test"@en .  
ex:ex4 exv:p "42"^^xsd:integer .  
ex:ex5 exv:p 42 .
```

```
SELECT * WHERE {  
    ?s exv:p 42 .  
}
```

Answer:

```
{ {?s ↦ ex:ex4}, {?s ↦ ex:ex5} }
```

Outline

1. Application Architecture
2. Basic Graph Patterns
3. Complex Graph Patterns
4. Navigational Graph Patterns: Property path
5. SPARQL Output Forms

Complex graph patterns

- From BGPs, we can use query algebra (e.g., SPARQL algebra) to form more complex queries, i.e., **complex graph patterns**.
- Operations to form complex graph patterns include projection, union, difference, joins, intersection, anti-join, left-join, etc.

Projection

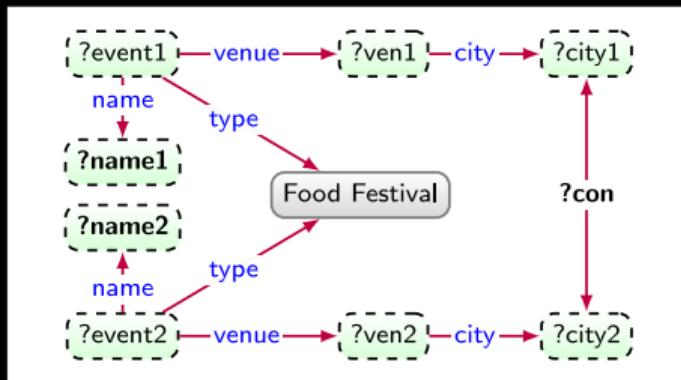
- Projection is realized in SPARQL via the SELECT output form.
- **SELECT *varlist*** returns a sequence of solution mappings restricted to the given variables in *varlist*.
 - If a variable in *varlist* does not occur in the graph pattern, it will be returned unbound.
 - If **all** variables in SELECT clause are unbound, the corresponding solution mapping is empty.
 - **SELECT *** returns solution mappings over all variables in the graph pattern.
 - **SELECT DISTINCT *varlist*** removes duplicate solution mappings.
 - **SELECT DISTINCT *** returns the whole multiset (set) of solution mappings over all variables in the graph pattern.

Projection example

Find the name of two food festivals that are held in cities connected (by any means) to each other. Return the names of the events as well as the kind of connections between the cities in which the two events are held.

Projection example

Find the name of two food festivals that are held in cities connected (by any means) to each other. Return the names of the events as well as the kind of connections between the cities in which the two events are held.



The projected variables are bold-printed.

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00"^^xsd:dateTime ;
  exv:end "2018-03-29T20:00:00"^^xsd:dateTime ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucia exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
    exv:name ?name1 ;
    exv:venue ?ven1 .

  ?event2 rdf:type exv:FoodFestival ;
    exv:name ?name2 ;
    exv:venue ?ven2 .

  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

Projection variables are specified in the SELECT clause.

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
    exv:name "Festival de Viña" ;
    exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
    exv:DrinksFestival ;
    exv:name "Ñam" ;
    exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
    exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
    exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
    exv:name "Food Truck"^^xsd:string ;
    exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
    exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
    exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
    exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
    ?event1 rdf:type exv:FoodFestival ;
        exv:name ?name1 ; exv:venue ?ven1 .
    ?event2 rdf:type exv:FoodFestival ;
        exv:name ?name2 ; exv:venue ?ven2 .
    ?ven1 exv:city ?city1 .
    ?ven2 exv:city ?city2 .
    ?city1 ?con ?city2 .
    ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
--------	------	--------

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
    exv:name "Festival de Viña" ;
    exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
    exv:DrinksFestival ;
    exv:name "Ñam" ;
    exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
    exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
    exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
    exv:name "Food Truck"^^xsd:string ;
    exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
    exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
    exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
    exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
    ?event1 rdf:type exv:FoodFestival ;
        exv:name ?name1 ; exv:venue ?ven1 .
    ?event2 rdf:type exv:FoodFestival ;
        exv:name ?name2 ; exv:venue ?ven2 .
    ?ven1 exv:city ?city1 .
    ?ven2 exv:city ?city2 .
    ?city1 ?con ?city2 .
    ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
    exv:name "Festival de Viña" ;
    exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
    exv:DrinksFestival ;
    exv:name "Ñam" ;
    exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
    exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
    exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
    exv:name "Food Truck"^^xsd:string ;
    exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
    exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
    exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
    exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
    ?event1 rdf:type exv:FoodFestival ;
        exv:name ?name1 ; exv:venue ?ven1 .
    ?event2 rdf:type exv:FoodFestival ;
        exv:name ?name2 ; exv:venue ?ven2 .
    ?ven1 exv:city ?city1 .
    ?ven2 exv:city ?city2 .
    ?city1 ?con ?city2 .
    ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
  exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
  exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
  exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
  exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
  exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
  exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam
Food Truck	exv:flight	Ñam

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
  exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
  exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
  exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam
Food Truck	exv:flight	Ñam
Food Truck	exv:flight	Ñam

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
  exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
  exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
  exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam
Food Truck	exv:flight	Ñam
Food Truck	exv:flight	Ñam
Ñam	exv:bus	Food Truck

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
  exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
  exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
  exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam
Food Truck	exv:flight	Ñam
Food Truck	exv:flight	Ñam
Ñam	exv:bus	Food Truck
Ñam	exv:flight	Food Truck

Projection example

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00^^xsd:dateTime" ;
  exv:end "2018-03-29T20:00:00^^xsd:dateTime" ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
  exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
  exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam
Food Truck	exv:flight	Ñam
Food Truck	exv:flight	Ñam
Ñam	exv:bus	Food Truck
Ñam	exv:flight	Food Truck
Ñam	exv:flight	Food Truck

Projection: SELECT DISTINCT

```
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival,
  exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00"^^xsd:dateTime ;
  exv:end "2018-03-29T20:00:00"^^xsd:dateTime ;
  exv:venue ex:SantaLucía .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
ex:PiscinaOlímpica exv:city ex:Arica .
ex:QuintaVergara exv:city ex:ViñadelMar .

ex:Santiago exv:bus ex:ViñadelMar ;
  exv:flight ex:ViñadelMar, ex:Arica .
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;
  exv:flight ex:Santiago .
ex:Arica exv:bus ex:ViñadelMar ;
  exv:flight ex:Santiago .
```

```
SELECT DISTINCT ?name1 ?con ?name2 WHERE {
  ?event1 rdf:type exv:FoodFestival ;
    exv:name ?name1 ; exv:venue ?ven1 .
  ?event2 rdf:type exv:FoodFestival ;
    exv:name ?name2 ; exv:venue ?ven2 .
  ?ven1 exv:city ?city1 .
  ?ven2 exv:city ?city2 .
  ?city1 ?con ?city2 .
  ?city2 ?con ?city1 .
}
```

?name1	?con	?name2
Food Truck	exv:bus	Food Truck
Food Truck	exv:bus	Ñam
Food Truck	exv:flight	Ñam
Ñam	exv:bus	Food Truck
Ñam	exv:flight	Food Truck

Union

- To express union between two graph patterns, SPARQL uses keyword UNION.
- The two graph patterns to be unioned are grouped using braces { ... }.
 - We write $\{P_1\}$ UNION $\{P_2\}$ to express the union of graph patterns P_1 and P_2
- Result in a **multiset union** of the answers of the two graph patterns.
- Identical variables within different UNION patterns do not influence each other.
- Some variables may be **unbound** when a graph pattern in the UNION pattern has a variable that does not occur in the other graph pattern.

Data

```
ex:EID13 rdf:type exv:TheatreFestival, exv:MusicFestival ;
  exv:name "Santiago a Mil" ;
  exv:venue ex:PlazadelaConstitución ;
  exv:start "2023-01-09T09:00:00"^^xsd:dateTime .
ex:EID14 rdf:type exv:MusicFestival ;
  exv:name "Festival de Viña" ;
  exv:venue ex:QuintaVergara .
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival, exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:venue ex:SantaLucía ;
  exv:start "2018-03-22T12:00:00"^^xsd:dateTime ;
  exv:end "2018-03-29T20:00:00"^^xsd:dateTime .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

ex:SantaLucía exv:city ex:Santiago .
ex:Sotomayor exv:city ex:ViñadelMar .
```

Data (cont.)

```
ex:PiscinaOlímpica exv:city ex:Arica .  
ex:QuintaVergara exv:city ex:ViñadelMar .  
ex:PlazadelaConstitución exv:city ex:Santiago .  
  
ex:Santiago exv:bus ex:ViñadelMar ;  
    exv:flight ex:ViñadelMar, ex:Arica .  
ex:ViñadelMar exv:bus ex:Arica, ex:Santiago ;  
    exv:flight ex:Santiago .  
ex:Arica exv:bus ex:ViñadelMar ;  
    exv:flight ex:Santiago .
```

Union example

List the name of all events that are held in either Santiago or Arica.
Indicate in the answer whether the events are held in Santiago or Arica.

Union example

List the name of all events that are held in either Santiago or Arica.
Indicate in the answer whether the events are held in Santiago or Arica.

```
SELECT ?name ?city WHERE {  
    ?event exv:venue ?ven ;  
           exv:name ?name .  
    {  
        ?ven exv:city ex:Santiago .  
    }  
UNION  
    {  
        ?ven exv:city ex:Arica .  
    }  
    ?ven exv:city ?city .  
}
```

Union example

List the name of all events that are held in either Santiago or Arica.
Indicate in the answer whether the events are held in Santiago or Arica.

```
SELECT ?name ?city WHERE {  
    ?event exv:venue ?ven ;  
            exv:name ?name .  
    {  
        ?ven exv:city ex:Santiago .  
    }  
UNION  
    {  
        ?ven exv:city ex:Arica .  
    }  
    ?ven exv:city ?city .  
}
```

?name	?city
Food Truck	ex:Arica
Ñam	ex:Santiago
Santiago a Mil	ex:Santiago

Union example

UNION can also result in unbound variables in the solution mappings. For example, for the query: “List the food festivals in Santiago and the music festivals in either Santiago or Viña del Mar. Separate the food festivals and the music festivals in different columns.”

Union example

UNION can also result in unbound variables in the solution mappings. For example, for the query: “List the food festivals in Santiago and the music festivals in either Santiago or Viña del Mar. Separate the food festivals and the music festivals in different columns.”

```
SELECT ?foodfest ?musicfest WHERE {  
  {  
    ?foodfest rdf:type exv:FoodFestival ;  
              exv:venue [ exv:city ex:Santiago ] .  
  }  
UNION  
{  
  ?musicfest rdf:type exv:MusicFestival ;  
              exv:venue ?ven .  
  { ?ven exv:city ex:Santiago }  
UNION  
  { ?ven exv:city ex:ViñadelMar }  
}  
}
```

Union example

UNION can also result in unbound variables in the solution mappings. For example, for the query: “List the food festivals in Santiago and the music festivals in either Santiago or Viña del Mar. Separate the food festivals and the music festivals in different columns.”

```
SELECT ?foodfest ?musicfest WHERE {
  {
    ?foodfest rdf:type exv:FoodFestival ;
               exv:venue [ exv:city ex:Santiago ] .
  }
  UNION
  {
    ?musicfest rdf:type exv:MusicFestival ;
               exv:venue ?ven .
    { ?ven exv:city ex:Santiago }
  UNION
    { ?ven exv:city ex:ViñadelMar }
  }
}
```

?foodfest	?musicfest
ex:EID15	
	ex:EID13
	ex:EID14

Optional

- OPTIONAL operator applies left-join between two graphs.
 - $P_1 \text{ OPTIONAL } \{ P_2 \}$ means: get all solution mappings for P_1 , and then **optionally** join with solution mappings of P_2 if any.
 - If a solution mapping for P_1 cannot be joined with any solution mapping for P_2 , then the solution mapping for P_1 is still returned.

Optional example

List the name of all food festivals and music festivals and optionally their start date/time."

Optional example

List the name of all food festivals and music festivals and optionally their start date/time."

```
SELECT ?name ?start WHERE {  
  { ?event rdf:type exv:FoodFestival . }  
UNION  
  { ?event rdf:type exv:MusicFestival . }  
  ?event exv:name ?name .  
}  
OPTIONAL  
{ ?event exv:start ?start . }  
}
```

Optional example

List the name of all food festivals and music festivals and optionally their start date/time."

```
SELECT ?name ?start WHERE {
  { ?event rdf:type exv:FoodFestival . }
  UNION
  { ?event rdf:type exv:MusicFestival . }
  ?event exv:name ?name .
}
OPTIONAL
{ ?event exv:start ?start . }
```

?name	?start
Food Truck	
Festival de Viña	
Santiago a Mil	2023-01-09T09:00:00.000Z
Ñam	2018-03-22T12:00:00.000Z

Notes on UNION- OPTIONAL combination

- OPTIONAL always applies to one pattern group, specified to the right of the OPTIONAL keyword.
- OPTIONAL and UNION has equal precedence. Grouping is left-associative.

Notes on UNION- OPTIONAL combination

- OPTIONAL always applies to one pattern group, specified to the right of the OPTIONAL keyword.
- OPTIONAL and UNION has equal precedence. Grouping is left-associative.

```
{ ?book ex:publishedBy <http://springer.com> .  
  { ?book ex:author ?author . } UNION  
  { ?book ex:editor ?author . } OPTIONAL  
  { ?author ex:surname ?name . } }
```

is equivalent to

Notes on UNION- OPTIONAL combination

- OPTIONAL always applies to one pattern group, specified to the right of the OPTIONAL keyword.
- OPTIONAL and UNION has equal precedence. Grouping is left-associative.

```
{ ?book ex:publishedBy <http://springer.com> .  
  { ?book ex:author ?author . } UNION  
  { ?book ex:editor ?author . } OPTIONAL  
  { ?author ex:surname ?name . } }
```

is equivalent to

```
{ ?book ex:publishedBy <http://springer.com> .  
  { { ?book ex:author ?author . } UNION  
    { ?book ex:editor ?author . }  
  } OPTIONAL { ?author ex:surname ?name . } }
```

Multiple OPTIONAL patterns

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
_:a foaf:name "Alice" ; foaf:homepage <http://work.example.org/alice/> .  
_:b foaf:name "Bob" ; foaf:mbox <mailto:bob@work.example> .
```

```
SELECT ?name ?mbox ?hpage  
WHERE { ?x foaf:name ?name .  
        OPTIONAL { ?x foaf:mbox ?mbox } .  
        OPTIONAL { ?x foaf:homepage ?hpage }  
}
```

Multiple OPTIONAL patterns

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
_:a foaf:name "Alice" ; foaf:homepage <http://work.example.org/alice/> .  
_:b foaf:name "Bob" ; foaf:mbox <mailto:bob@work.example> .
```

```
SELECT ?name ?mbox ?hpage  
WHERE { ?x foaf:name ?name .  
        OPTIONAL { ?x foaf:mbox ?mbox } .  
        OPTIONAL { ?x foaf:homepage ?hpage }  
}
```

name	mbox	hpage
Alice		<http://work.example.org/alice/>
Bob	<mailto:bob@work.example>	

Why Filters?

Even with complex query patterns, some queries are not expressible:

- “Which persons are between 18 and 23 years old?”
- “Which person has a name that contains a hyphen character?”
- “List the English name of the capital city of European countries”.

We use **filter** as a general mechanism for such expressions.

Filter

- Syntax: FILTER(filterExpression)
- By instantiating its variables, a filter expression returns an **effective boolean value** (**true** or **false**), or produces an error.
 - See <https://www.w3.org/TR/sparql11-query/#ebv>.
- Evaluation: eliminate a solution mapping if instantiating variables in the filter according to the solution mapping results in the EBV **false** or produces an error.
- Many SPARQL filters come from outside RDF, e.g., XQuery/XPath
- Filter can be used to express some form of negation.
- <https://www.w3.org/TR/sparql11-query/#expressions>.

Example

```
SELECT ?book
WHERE {
    ?book ex:publishedBy <http://springer.com> .
    ?book ex:price ?price
    FILTER (?price < 35)
}
```

Above, any solution mapping where the value of ?price is less than 35 is eliminated from the result.

Filter: SPARQL Boolean Operators



Unary Boolean operators: ! ~ ~ !A is **true** if A is **false**, vice versa.

Logical connectives: || , &&

Comparison operators: <, =, >, <=, >=, !=

- Comparison for literals according to the natural ordering
- Support for numerical datatypes (xsd:integer, xsd:decimal, etc.), xsd:dateTime, xsd:string (alphabetical order), xsd:Boolean ($1 > 0$)
- For non-literals, only = and != are available.
- Comparison cannot be done between incompatible types, e.g., between an xsd:string literal and an xsd:integer literal.

Filter: Arithmetic Functions

Unary functions: +, -

Binary functions: +, -, *, /

- Support for numerical datatypes.
- Not Boolean; used to obtain a value from other values in filter expression. For example:

```
FILTER( ?weight / (?size * ?size) >= 25 )
```

Other Functions and Function Forms

var is a variable, expr₁, expr₂, expr₃ are expressions interpreted as an EBV, term, term₁, term₂ are RDF terms (IRIs, literals, blank nodes), pattern is a graph pattern, lit is a literal, res is an IRI

BOUND(var)	true if var is a bound variable
IF(expr ₁ , expr ₂ , expr ₃)	returns EBV of expr ₂ if expr ₁ is true , otherwise returns EBV of expr ₃
EXISTS \{ pattern \}	true if pattern matches; false otherwise
NOT EXISTS \{ pattern \}	false if pattern matches; true otherwise
sameTerm(term ₁ , term ₂)	true if term ₁ and term ₂ are the same; false otherwise. more general than = operator
term IN (expr ₁ , ...)	true if term can be found in the list on the right hand side
term NOT IN (expr ₁ , ...)	true if term cannot be found in the list
isIRI(term), isURI(term)	true if term is an IRI
isBlank(term)	true if term is a blank node
isLiteral(term)	true if term is a literal
isNumeric(term)	true if term is a numeric value 17 and "17"\^\^xsd:integer" are numeric, while "17" is not

Other Functions and Function Forms

STR(lit)	returns the lexical form of the literal <code>lit</code> .
STR(res)	returns the codepoint/string representation of the IRI <code>res</code>
LANG(lit)	returns the language tag of the literal <code>lit</code> , if any; returns "" otherwise
DATATYPE(lit)	returns the datatype of <code>lit</code>
IRI(lit), IRI(res)	returns an IRI from the literal <code>lit</code> or an IRI <code>res</code> <code>lit</code> must be a simple literal (without explicit datatype).
BNODE(), BNode(lit)	creates a blank node; if given a simple literal argument, the same literal within an expression for the same solution mapping yields the same blank node
STRDT(lit, res)	creates a typed literal with lexical form <code>list</code> and datatype <code>res</code>
STRLANG(lit, ltag)	creates a language-tagged literal with lexical form <code>list</code> and language tag <code>ltag</code>
UUID()	returns a fresh IRI using URN scheme (Note: not a HTTP IRI!)
STRUUID()	returns a string that is a scheme specific part of a UUID

Other Functions and Function Forms

- String functions: langMatches, REGEX, REPLACE, CONCAT, STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, STRENDs, CONTAINS, STRBEFORE, STRAFTER, ENCODE_FOR_URI
- Numeric functions: ABS, ROUND, CEIL, floor, RAND
- Data/Time functions: NOW, YEAR, MONTH, DAY, HOURS, MINUTES, SECONDS, TIMEZONE, TZ
- Hash functions: MD5, SHA1, SHA256, SHA384, SHA512
- Casting operations: STR, BOOL, DBL, FLT, DEC, INT, dT, ltrl

Scope of filters

```
{ ?x foaf:name ?name .  
?x foaf:mbox ?mbox .  
FILTER regex(?name, "Smith")  
}
```

```
{ FILTER regex(?name, "Smith")  
?x foaf:name ?name .  
?x foaf:mbox ?mbox .  
}
```

```
{ ?x foaf:name ?name .  
FILTER regex(?name, "Smith")  
?x foaf:mbox ?mbox .  
}
```

- Patterns can be grouped using pairs of braces.
- Filter is applied to the whole group in which the filter expression appears.
- The 3 patterns on the left have the same answers.

Filters in OPTIONAL patterns

```
@prefix : <http://example.org/book/> .  
@prefix ns: <http://example.org/ns#> .  
:book1 ns:title "SPARQL Tutorial" ; ns:price 42 .  
:book2 ns:title "The Semantic Web" ; ns:price 23 .
```

```
SELECT ?title ?price  
WHERE { ?x ns:title ?title .  
        OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }  
 }
```

Filters in OPTIONAL patterns

```
@prefix : <http://example.org/book/> .  
@prefix ns: <http://example.org/ns#> .  
:book1 ns:title "SPARQL Tutorial" ; ns:price 42 .  
:book2 ns:title "The Semantic Web" ; ns:price 23 .
```

```
SELECT ?title ?price  
WHERE { ?x ns:title ?title .  
        OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }  
}
```

title	price
SPARQL Tutorial	
The Semantic Web	23

Negation Using FILTER

Data:

```
[] foaf:name "Alice".  
[ foaf:name "Bob" ; foaf:age "35"^^xsd:integer ] .
```

Query:

```
SELECT ?name WHERE {  
    ?x foaf:name ?name .  
    OPTIONAL { ?x foaf:age ?age } .  
    FILTER (!bound(?age))  
}
```

returns

Negation Using FILTER

Data:

```
[] foaf:name "Alice".  
[ foaf:name "Bob" ; foaf:age "35"^^xsd:integer ] .
```

Query:

```
SELECT ?name WHERE {  
    ?x foaf:name ?name .  
    OPTIONAL { ?x foaf:age ?age } .  
    FILTER (!bound(?age))  
}
```

returns "Alice" as the only value for ?name

Testing for the presence of patterns

```
@prefix : <http://example.org/data/> .  
:alice rdf:type foaf:Person .  
:alice foaf:name "Alice" .  
:bob rdf:type foaf:Person .
```

Query:

```
SELECT ?person WHERE {  
    ?person rdf:type foaf:Person .  
    FILTER EXISTS { ?person foaf:name ?name }  
}
```

Answer:

Testing for the presence of patterns

```
@prefix : <http://example.org/data/> .  
:alice rdf:type foaf:Person .  
:alice foaf:name "Alice" .  
:bob rdf:type foaf:Person .
```

Query:

```
SELECT ?person WHERE {  
    ?person rdf:type foaf:Person .  
    FILTER EXISTS { ?person foaf:name ?name }  
}
```

Answer:

person

<http://example.org/data/alice>

Testing for the absence of patterns

```
@prefix : <http://example.org/data/> .  
:alice rdf:type foaf:Person .  
:alice foaf:name "Alice" .  
:bob rdf:type foaf:Person .
```

Query:

```
SELECT ?person WHERE {  
    ?person rdf:type foaf:Person .  
    FILTER NOT EXISTS { ?person foaf:name ?name }  
}
```

Answer:

Testing for the absence of patterns

```
@prefix : <http://example.org/data/> .  
:alice rdf:type foaf:Person .  
:alice foaf:name "Alice" .  
:bob rdf:type foaf:Person .
```

Query:

```
SELECT ?person WHERE {  
    ?person rdf:type foaf:Person .  
    FILTER NOT EXISTS { ?person foaf:name ?name }  
}
```

Answer:

person

<http://example.org/data/bob>

Removing possible solutions

```
@prefix : <http://example.org/data/> .  
:alice foaf:givenName "Alice" ; foaf:familyName "Smith" .  
:bob foaf:givenName "Bob" ; foaf:familyName "Jones" .  
:carol foaf:givenName "Carol" ; foaf:familyName "Smith" .
```

```
SELECT DISTINCT ?s WHERE {  
    ?s ?p ?o .  
    MINUS { ?s foaf:givenName "Bob" . }  
}
```

Answer:

Removing possible solutions

```
@prefix : <http://example.org/data/> .  
:alice foaf:givenName "Alice" ; foaf:familyName "Smith" .  
:bob foaf:givenName "Bob" ; foaf:familyName "Jones" .  
:carol foaf:givenName "Carol" ; foaf:familyName "Smith" .
```

```
SELECT DISTINCT ?s WHERE {  
    ?s ?p ?o .  
    MINUS { ?s foaf:givenName "Bob" . }  
}
```

Answer:

s

<http://example.org/data/carol>
<http://example.org/data/alice>

FILTER NOT EXISTS versus MINUS

- FILTER NOT EXISTS corresponds to testing whether a pattern exists in the data.
 - It works by examining the solution mappings/bindings already determined by the query pattern.
- MINUS removes matches based on evaluation of two patterns (like minus operation in sets).
 - In $P_1 \text{ MINUS } P_2$, P_2 can only remove matches in P_1 if P_1 and P_2 share some variables.

FILTER NOT EXISTS versus MINUS: Shared variables

```
@prefix : <http://example.org/data/> .  
:alice :likes :bob .
```

```
SELECT * {  
?s ?p ?o .  
FILTER NOT EXISTS { ?x ?y ?z . }  
}
```

Answer:

```
SELECT * {  
?s ?p ?o .  
MINUS { ?x ?y ?z . }  
}
```

Answer:

FILTER NOT EXISTS versus MINUS: Shared variables

```
@prefix : <http://example.org/data/> .  
:alice :likes :bob .
```

```
SELECT * {  
?s ?p ?o .  
FILTER NOT EXISTS { ?x ?y ?z . }  
}
```

Answer:

s p o

```
SELECT * {  
?s ?p ?o .  
MINUS { ?x ?y ?z . }  
}
```

Answer:

FILTER NOT EXISTS versus MINUS: Shared variables

```
@prefix : <http://example.org/data/> .  
:alice :likes :bob .
```

```
SELECT * {  
?s ?p ?o .  
FILTER NOT EXISTS { ?x ?y ?z . }  
}
```

Answer:

s p o

```
SELECT * {  
?s ?p ?o .  
MINUS { ?x ?y ?z . }  
}
```

Answer:

s p o

:alice :likes :bob

FILTER NOT EXISTS versus MINUS: Fixed pattern

```
@prefix : <http://example.org/data/> .  
:alice :likes :bob .
```

```
SELECT * {  
?s ?p ?o .  
FILTER  
NOT EXISTS { :alice :likes :bob . }  
}
```

Answer:

```
SELECT * {  
?s ?p ?o .  
MINUS {  
:alice :likes :bob . }  
}
```

Answer:

FILTER NOT EXISTS versus MINUS: Fixed pattern

```
@prefix : <http://example.org/data/> .  
:alice :likes :bob .
```

```
SELECT * {  
?s ?p ?o .  
FILTER  
NOT EXISTS { :alice :likes :bob . }  
}
```

Answer:

s p o

```
SELECT * {  
?s ?p ?o .  
MINUS {  
:alice :likes :bob . }  
}
```

Answer:

FILTER NOT EXISTS versus MINUS: Fixed pattern

```
@prefix : <http://example.org/data/> .  
:alice :likes :bob .
```

```
SELECT * {  
?s ?p ?o .  
FILTER  
NOT EXISTS { :alice :likes :bob . }  
}
```

Answer:

s	p	o

```
SELECT * {  
?s ?p ?o .  
MINUS {  
:alice :likes :bob . }  
}
```

Answer:

s	p	o
:alice	:likes	:bob

FILTER NOT EXISTS versus MINUS: Inner filters

@prefix : <http://example.org/data/> .

```
:ann :testA 70 .  
:ann :testB 70 .  
:ann :testB 80 .  
  
:bob :testA 80.5 .  
:bob :testB 90.5 .  
:bob :testB 95.0 .
```

```
SELECT * WHERE {  
    ?x :testA ?n  
    FILTER NOT EXISTS {  
        ?x :testB ?m .  
        FILTER(?n = ?m)  
    }  
}
```

Answer:

FILTER NOT EXISTS versus MINUS: Inner filters

@prefix : <http://example.org/data/> .

```
:ann :testA 70 .  
:ann :testB 70 .  
:ann :testB 80 .  
  
:bob :testA 80.5 .  
:bob :testB 90.5 .  
:bob :testB 95.0 .
```

```
SELECT * WHERE {  
    ?x :testA ?n  
    FILTER NOT EXISTS {  
        ?x :testB ?m .  
        FILTER(?n = ?m)  
    }  
}
```

Answer:

x	n
<http://example.org/data/b>	80.5

FILTER NOT EXISTS versus MINUS: Inner filters

@prefix : <http://example.org/data/> .

```
:ann :testA 70 .  
:ann :testB 70 .  
:ann :testB 80 .  
  
:bob :testA 80.5 .  
:bob :testB 90.5 .  
:bob :testB 95.0 .
```

```
SELECT * WHERE {  
  ?x :testA ?n  
  MINUS {  
    ?x :testB ?m .  
    FILTER(?n = ?m)  
  }  
}
```

Answer:

FILTER NOT EXISTS versus MINUS: Inner filters

@prefix : <http://example.org/data/> .

```
:ann :testA 70 .  
:ann :testB 70 .  
:ann :testB 80 .  
  
:bob :testA 80.5 .  
:bob :testB 90.5 .  
:bob :testB 95.0 .
```

```
SELECT * WHERE {  
    ?x :testA ?n  
    MINUS {  
        ?x :testB ?m .  
        FILTER(?n = ?m)  
    }  
}
```

Answer:

x	n
<http://example.org/data/b>	80.5
<http://example.org/data/a>	70

Sorting Results with ORDER BY

```
SELECT ?book, ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price
```

- Sorting as with comparison operators in filters.
- IRIs are sorted alphabetically.
- Ordering of elements of different types:
unbound variables < blank nodes < IRIs < RDF literals
- Spec does not define all possible orderings.
- Descending order: use ORDER BY DESC (?price)
- Ascending order (default): ORDER BY ASC (?price)
- Hierarchical ordering criteria: ORDER BY ASC(?price), title

LIMIT, OFFSET, and DISTINCT

- SELECT DISTINCT: removal of duplicates
- LIMIT: maximal number of results
- OFFSET: position of the first returned result (within the whole result).
- LIMIT and OFFSET only meaningful with ORDER BY.

```
SELECT DISTINCT ?book, ?price
WHERE { ?book <http://ex.org/price> ?price . }
ORDER BY ?price LIMIT 5 OFFSET 25
```

Assignment of New Values

Inside SELECT clause:

```
SELECT ?Item (?Pr * 1.1 AS ?NewP )  
WHERE { ?Item ex:price ?Pr . }
```

Note: cannot assign values to variables inside the expression.

Data:

```
ex:lemonade1 ex:price 3 .  
ex:icetea1 ex:price 3 .  
ex:coke1 ex:price 3.50 .  
ex:coffee1 ex:price "n/a" .
```

Result (leaves errors unbound):

Item	NewP
ex:lemonade1	3.3
ex:icetea1	3.3
ex:coke1	3.85
ex:coffee1	

Assignment of New Values (cont.)

Alternatively, using BIND:

```
SELECT ?Item ?NewP
WHERE { ?Item ex:price ?Pr .
        BIND (?Pr * 1.1 AS ?NewP) }
```

Data:

```
ex:lemonade1 ex:price 3 .
ex:icetea1 ex:price 3.
ex:coke1 ex:price 3.50 .
ex:coffee1 ex:price "n/a" .
```

Result (leaves errors unbound):

Item	NewP
ex:lemonade1	3.3
ex:icetea1	3.3
ex:coke1	3.85
ex:coffee1	

Assignment of New Values (cont.)

Note: BIND is evaluated **in-place**!

```
SELECT ?Item ?NewP
WHERE { BIND (?Pr * 1.1 AS ?NewP)
         ?Item ex:price ?Pr . }
```

Data:

```
ex:lemonade1 ex:price 3 .
ex:icetea1 ex:price 3.
ex:coke1 ex:price 3.50 .
ex:coffee1 ex:price "n/a" .
```

Result is empty:

Item	NewP
------	------

Providing Inline Data with VALUES

```
:drink1 rdfs:label "Latte" ; ex:price 4 .  
:drink2 rdfs:label "Capuccino" ; ex:price 3.5 .  
:drink3 rdfs:label "Dark Roast" ; ex:price 2 .  
:drink4 rdfs:label "Espresso" ; ex:price 3.5 .
```

Providing Inline Data with VALUES

```
:drink1 rdfs:label "Latte" ; ex:price 4 .  
:drink2 rdfs:label "Capuccino" ; ex:price 3.5 .  
:drink3 rdfs:label "Dark Roast" ; ex:price 2 .  
:drink4 rdfs:label "Espresso" ; ex:price 3.5 .
```

Use VALUES to enumerate tuples of values to be assigned to variables.

```
SELECT ?drink ?name ?price  
WHERE {  
    ?drink rdfs:label ?name ;  
            ex:price ?price .  
    VALUES (?drink ?name)  
    { (UNDEF "Latte")  
     (:drink2 UNDEF)  
     (:drink5 "Espresso")  
    }  
}
```

Providing Inline Data with VALUES

```
:drink1 rdfs:label "Latte" ; ex:price 4 .  
:drink2 rdfs:label "Capuccino" ; ex:price 3.5 .  
:drink3 rdfs:label "Dark Roast" ; ex:price 2 .  
:drink4 rdfs:label "Espresso" ; ex:price 3.5 .
```

Use VALUES to enumerate tuples of values to be assigned to variables.

```
SELECT ?drink ?name ?price  
WHERE {  
    ?drink rdfs:label ?name ;  
            ex:price ?price .  
VALUES (?drink ?name)  
{ (UNDEF "Latte")  
  (:drink2 UNDEF)  
  (:drink5 "Espresso")  
}  
}
```

drink	name	price
:drink1	Latte	4
:drink2	Capuccino	3.5

Aggregates

Count items:

```
SELECT (COUNT(?Item) AS ?C)
WHERE { ?Item ex:price ?Pr . }
```

Data:

```
ex:smoothie1 ex:price 4 ;
          a ex:Colddrink .
ex:icetea1 ex:price 3 ;
          a ex:Colddrink .
ex:coke1    ex:price 3.50 ;
          a ex:Colddrink .
ex:tea1     ex:price 3 ;
          a ex:Hotdrink .
ex:coffee1  ex:price "n/a" ;
          a ex:Hotdrink .
```

Results:

?C
5

Aggregates (cont.)

Count categories:

```
SELECT (COUNT(?Ty) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
```

Data:

```
ex:smoothie1 ex:price 4 ;
            a ex:Colddrink .
ex:icetea1 ex:price 3 ;
            a ex:Colddrink .
ex:coke1    ex:price 3.50 ;
            a ex:Colddrink .
ex:tea1     ex:price 3 ;
            a ex:Hotdrink .
ex:coffee1   ex:price "n/a" ;
            a ex:Hotdrink .
```

Results:

?C
5

Aggregates (cont.)

Count distinct categories:

```
SELECT (COUNT(DISTINCT ?Ty) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
```

Data:

```
ex:smoothie1 ex:price 4 ;
           a ex:Colddrink .
ex:icetea1 ex:price 3 ;
           a ex:Colddrink .
ex:coke1   ex:price 3.50 ;
           a ex:Colddrink .
ex:tea1    ex:price 3 ;
           a ex:Hotdrink .
ex:coffee1 ex:price "n/a" ;
           a ex:Hotdrink .
```

Results:

?C
2

Aggregates with Grouping

Count item per categories:

```
SELECT ?Ty (COUNT(?Item) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
GROUP BY ?Ty
```

Data:

```
ex:smoothie1 ex:price 4 ;
          a ex:Colddrink .
ex:icetea1 ex:price 3 ;
          a ex:Colddrink .
ex:coke1    ex:price 3.50 ;
          a ex:Colddrink .
ex:tea1     ex:price 3 ;
          a ex:Hotdrink .
ex:coffee1   ex:price "n/a" ; a ex:Hotdrink .
```

Results:

Ty	C
ex:Colddrink	3
ex:Hotdrink	2

Filtering Groups

Count item per categories, for those categories with more than two items:

```
SELECT ?Ty (COUNT(?Item) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
GROUP BY ?Ty
HAVING COUNT(?Item) > 2
```

Data:

```
ex:smoothie1 ex:price 4 ;
          a ex:Colddrink .
ex:icetea1 ex:price 3 ;
          a ex:Colddrink .
ex:coke1    ex:price 3.50 ;
          a ex:Colddrink .
ex:tea1     ex:price 3 ; a ex:Hotdrink .
ex:coffee1  ex:price "n/a" ; a ex:Hotdrink .
```

Results:

?Ty	?C
ex:Colddrink	3

Other Aggregates

- SUM(?X)
- AVG(?X)
- MIN(?X)
- MAX(?X)
- GROUP_CONCAT(?X ; separator="|") – concatenate values with a given separator string ‘|’
- SAMPLE(?X) – ‘pick’ one non-deterministically

Subqueries

Subqueries: SELECT query inside a graph pattern.

“List all distinct titles of papers authored by at most 6 co-authors of Pascal Hitzler”

```
PREFIX swp: <http://data.semanticweb.org/person/>
SELECT DISTINCT ?title
WHERE {
  ?paper foaf:maker ?person ; rdfs:label ?title .
  { SELECT DISTINCT ?person
    WHERE {
      ?doc foaf:maker swp:pascal-hitzler, ?person .
      FILTER (?person != swp:pascal-hitzler)
    } LIMIT 6
  }
}
```

Outline

1. Application Architecture
2. Basic Graph Patterns
3. Complex Graph Patterns
4. Navigational Graph Patterns: Property path
5. SPARQL Output Forms

Property Path Expressions

Allows one to query using arbitrary length of paths in the graphs.

“List all names of people who transitively co-authors with Pascal Hitzler”

```
PREFIX swp: <http://data.semanticweb.org/person/>
SELECT DISTINCT ?name
WHERE {
    swp:pascal-hitzler (^foaf:maker/foaf:maker)+/foaf:name ?name
}
```

That is, we find the name of:

1. people who co-authors with Pascal Hitzler;
2. people who co-authors with the people from (1)
3. people who-co-authors with the people from (2)
4. etc.

Property Path Syntax Forms

The forms are somewhat similar to regular expression.

1. iri - an IRI, a path of length one.
2. \wedge path - inverse of path
3. path₁ / path₂ - concatenation of path₁ followed by path₂
4. path₁ | path₂ - alternative between path₁ and path₂ (try all possibilities)
5. path* - zero or more concatenation of path
6. path+ - one or more concatenation of path
7. path? - zero or one of path
8. !(iri₁|...|iri_n) - an IRI not one of iri₁,..., iri_n.
9. !(\wedge iri₁|...| \wedge iri_n) - an IRI not one of reverse of iri₁,..., iri_n. Can be combined with the negated path expression in (8)
10. (path) - grouping of path with brackets to control precedence

Precedence from highest to lowest: IRI, negated property sets, groups, unary operators, unary inverse links, concatenation binary operator, binary operator for alternatives

Property path: Examples (only the graph pattern)

```
{ :book1 dc:title|rdfs:label ?displayString }
```

is equivalent to

```
{
  { :book1 dc:title ?displayString }
  UNION
  { :book1 rdfs:label ?displayString }
}
```

Property path: Examples (only the graph pattern)

```
{  
  ?x foaf:mbox <mailto:alice@example> .  
  ?x foaf:knows/foaf:name ?name .  
}
```

is equivalent to

```
{  
  ?x foaf:mbox <mailto:alice@example> .  
  ?x foaf:knows [ foaf:name ?name ] .  
}
```

Property path: Examples (only the graph pattern)

```
{  
  ?x foaf:mbox <mailto:alice@example> .  
  ?x foaf:knows/foaf:knows/foaf:name ?name .  
}
```

is equivalent to

```
{  
  ?x foaf:mbox <mailto:alice@example> .  
  ?x foaf:knows [ foaf:knows [ foaf:name ?name ] ].  
}
```

Property path: Examples (only the graph pattern)

Someone Alice knows may well know Alice herself. To filter out Alice from the output:

```
{  
    ?x foaf:mbox <mailto:alice@example> .  
    ?x foaf:knows/foaf:knows ?y .  
    FILTER (?x != ?y)  
    ?y foaf:name ?name .  
}
```

Property path: Examples (only the graph pattern)

```
{  
  ?x foaf:mbox <mailto:alice@example>  
}
```

is equivalent to

```
{  
  <mailto:alice@example> ^foaf:mbox ?x .  
}
```

Property path: Examples (only the graph pattern)

```
{  
    ?x foaf:knows/^foaf:knows ?y .  
    FILTER(?x != ?y)  
}
```

is equivalent to

```
{  
    ?x foaf:knows ?gen1 .  
    ?y foaf:knows ?gen1 .  
    FILTER(?x != ?y)  
}
```

Property path: Examples (only the graph pattern)

Find the names of all people that can be reached from Alice by foaf:knows

```
{  
  ?x foaf:mbox <mailto:alice@example> .  
  ?x foaf:knows+/foaf:name ?name .  
}
```

Property path: Examples (only the graph pattern)

Get all ancestors of Alice.

```
{  
    ?ancestor (ex:motherOf|ex:fatherOf)+ ex:alice .  
}
```

Find connected nodes, but not by rdf:type in either direction.

```
{  
    ?x !(rdf:type|^rdf:type) ?y .  
}
```

Outline

1. Application Architecture
2. Basic Graph Patterns
3. Complex Graph Patterns
4. Navigational Graph Patterns: Property path
5. SPARQL Output Forms

Output Form SELECT

- SELECT returns sequence of solution mappings.
- Syntax: SELECT `variableList` or SELECT *
- **Advantage:** simple sequential processing of results.
- **Disadvantage:** structure and relationships between the objects are lost.

Output Form CONSTRUCT

- CONSTRUCT returns an RDF graph (i.e., a set of triples) created using results from the graph patterns.
- Can be used to transform a graph to another.
- **Advantage:** structured results data between the objects
- **Disadvantage:** harder to process sequentially
- **Disadvantage:** if a solution mapping contains unbound variable, triples corresponding to that solution mapping will be omitted.

PREFIX ex: <http://example.org/>

```
CONSTRUCT {  
    ?person ex:mailbox ?email .  
    ?person ex:telephone ?tel . }  
WHERE {  
    ?person ex:email ?email .  
    ?person ex:tel ?tel . }
```

CONSTRUCT Template with Blank Nodes

Given data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
_:a foaf:firstname "Alice" ; foaf:surname "Hacker" .  
_:b foaf:firstname "Bob" ; foaf:surname "Hacker" .
```

and query:

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>  
CONSTRUCT {  
    ?x vcard:N _:v .  
    _:v vcard:givenName ?gname ; vcard:familyName ?fname  
} WHERE {  
    ?x foaf:firstname ?gname .  
    ?x foaf:surname ?fname }
```

CONSTRUCT Template with Blank Nodes (cont.)



we would obtain an RDF graph:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
_:v1 vcard:N _:x .  
_:x vcard:givenName "Alice" ;  
  vcard:familyName "Hacker" .  
_:v2 vcard:N _:z .  
_:z vcard:givenName "Bob" ;  
  vcard:familyName "Hacker" .
```

Notice that the blank nodes in the output may have completely different IDs than what was provided by the solution mappings and the template.

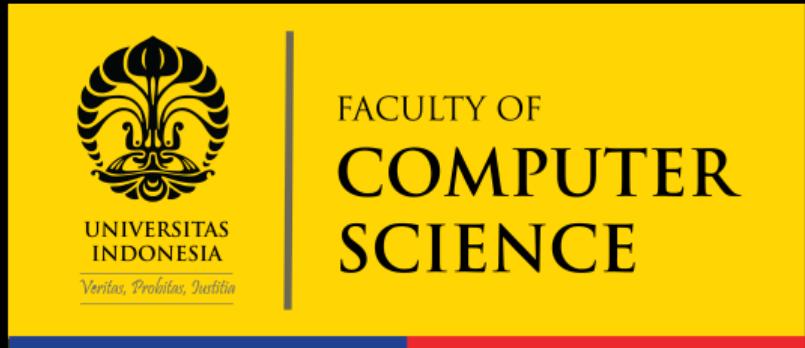
ASK and DESCRIBE Output Forms

- ASK: checks if the query has at least one answer, i.e., non-empty solution – returns true/false.
- DESCRIBE: returns an RDF description for each resulting IRI – the actual description returned is application-dependent.

```
DESCRIBE ?x WHERE { ?x <http://ex.org/emplID> "123" }
```

may return something like (depending on the triple store configuration):

```
_:a ex0rg:emplID "123" ;
      foaf:mbox_sha1sum "ABCD1234" ;
      vcard:N
          [ vcard:Family "Smith" ;
            vcard:Given "John" ] .
foaf:mbox_sha1sum a owl:InverseFunctionalProperty .
```



Semantic Web 05: Semantic Schema with RDF Schema

Adila Krisnadhi – adila@cs.ui.ac.id Faculty of Computer Science, Universitas
Indonesia

Outline

1. Syntax and Semantic

2. RDF Schema

Syntax vs. Semantics

- **Syntax**
 - Collection of symbols/terms accompanied by rules (i.e., grammar) that govern how sentences/statements in a language are formed of those symbols/terms.
 - Collection of terms = **vocabulary**
- **Semantics (logic-based)**
 - Meaning of those symbols and statements
 - **Interpretation** specifies what each symbol and statement stand for and when a statement becomes true or false.
 - Often expressed using mathematical sets, relations, etc. to make it unambiguous.

Entailment

- **Entailment:** Relationship between sentences that hold whenever a set of sentences logically follows from other sets of sentences.

Entailment

- **Entailment:** Relationship between sentences that hold whenever a set of sentences logically follows from other sets of sentences.
- A set of statements G **entails** another set of statements H iff the following holds: whenever all statements in G are true, then all statements in H are also true.

Entailment

- **Entailment:** Relationship between sentences that hold whenever a set of sentences logically follows from other sets of sentences.
- A set of statements G **entails** another set of statements H iff the following holds: whenever all statements in G are true, then all statements in H are also true.
- **Reasoning/inference:** Given two sets of statements G and H , if we assume that all statements in G are true, **decide** whether all statements in H are also true.

Entailment: Example in first-order logic

The set $\{Human(Socrates), \forall x.(Human(x) \rightarrow Mortal(x))\}$

Entailment: Example in first-order logic

The set $\{Human(Socrates), \forall x.(Human(x) \rightarrow Mortal(x))\}$
entails $Mortal(Socrates)$

Entailment: Example in first-order logic

The set $\{Human(Socrates), \forall x.(Human(x) \rightarrow Mortal(x))\}$ entails $Mortal(Socrates)$

- Terms: *Socrates* (constant) and *x* (variable).
- Statements: $Human(Socrates)$, $Human(x)$, $Mortal(x)$, and $\forall x.(Human(x) \rightarrow Mortal(x))$
- Interpretation and entailment? (see whiteboard)

Outline

1. Syntax and Semantic

2. RDF Schema

IRI prefixes

RDF semantics specification: <https://www.w3.org/TR/rdf11-mt/>

We use the following IRI prefixes.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <http://www.example.org/data/> .
@prefix o: <http://www.example.org/vocab#>.
```

Semantics of RDF graph

- RDF graph is just a set of triples
 - no description about shape/structure governing the graph,
 - no specific meaning assigned to any IRIs – meaning of datatype IRIs are not defined by RDF semantics.

Semantics of RDF graph

- RDF graph is just a set of triples
 - no description about shape/structure governing the graph,
 - no specific meaning assigned to any IRIs – meaning of datatype IRIs are not defined by RDF semantics.
- When is an RDF graph true?

Semantics of RDF graph

- RDF graph is just a set of triples
 - no description about shape/structure governing the graph,
 - no specific meaning assigned to any IRIs – meaning of datatype IRIs are not defined by RDF semantics.
- When is an RDF graph true? Answer: when all of its triples are true.

Semantics of RDF graph

- RDF graph is just a set of triples
 - no description about shape/structure governing the graph,
 - no specific meaning assigned to any IRIs – meaning of datatype IRIs are not defined by RDF semantics.
- When is an RDF graph true? Answer: when all of its triples are true.
- When is a triple true?

RDF semantics (cont.)

- A triple (s, p, o) is true iff there is a binary relation identified by p connecting entity identified by s to an entity identified by o .
 - Since one can almost always create such a binary relation, a triple is almost always true.
 - If o is a literal, then the lexical form must be compatible with its datatype for the triple to be true. For example, "test"^^xsd:integer is ill-typed.

RDF semantics (cont.)

- A triple (s, p, o) is **true** iff there is a binary relation identified by p connecting entity identified by s to an entity identified by o .
 - Since one can almost always create such a binary relation, a triple is almost always true.
 - If o is a literal, then the lexical form must be compatible with its datatype for the triple to be true. For example, "test"^^xsd:integer is ill-typed.
- An RDF graph (i.e., a set of RDF triples) is **true** iff all triples in the graph are true.

Simple graph entailment

xxx ppp yyy .

entails

xxx ppp yyy .

Simple graph entailment

xxx ppp yyy .

entails

xxx ppp yyy .

xxx ppp yyy .

entails

xxx ppp _:nnn .

where _:nnn must be
a **new** blank node in
the graph.

Simple graph entailment

xxx ppp yyy .

entails

xxx ppp yyy .

xxx ppp yyy .

entails

xxx ppp _:nnn .

where _:nnn must be
a **new** blank node in
the graph.

xxx ppp yyy .

entails

_:nnn ppp yyy .

where _:nnn must be
a **new** blank node in
the graph.

RDF entailment

```
xxx ppp "aaa"^^ddd .
```

entails

```
xxx ppp _:nnn .  
_:nnn rdf:type ddd .
```

where _:nnn must be a new
blank node in the graph.

RDF entailment

```
xxx ppp "aaa"^^ddd .
```

entails

```
xxx ppp _:nnn .  
_:nnn rdf:type ddd .
```

where _:nnn must be a new
blank node in the graph.

```
xxx ppp yyy .
```

entails

```
ppp rdf:type rdf:Property .
```

RDF Schema

- RDF Schema (RDFS) defines meaning for some particular IRIs to allow us to describe structures in the graph.
- RDFS does not introduce new syntax — everything is syntactically described in RDF.

RDFS features

- Core vocabulary to model classes, instances, property constraints, class and property hierarchy.
- Inference rules for reasoning.
- Auxiliary vocabulary with partial formal meaning for modeling container classes and properties, collections, reification, and additional utility.

rdf:type and rdfs:Class

```
1 | :Surabaya rdf:type o:City .  
2 | :Jakarta a o:City .  
3 | o:City rdf:type rdfs:Class .
```

- `rdf:type` = instance-of relation = set-membership relation.
- Line 1 reads “:Surabaya is a o:City.” Line 2 reads “:Jakarta is a o:City.”
- o:City is a class – set of individuals (in this case cities).
- RDFS semantic implies that o:City must be an instance of `rdfs:Class`. Thus, line 3 is actually implied by line 1 (or line 2).
- `rdfs:Class` is a class that contains all classes, including itself.

rdfs:subClassOf

```
1 | :UI rdf:type o:University .  
2 | :Pertamina a o:OilCompany .  
3 | o:University rdfs:subClassOf o:EducationInstitution .  
4 | o:OilCompany rdfs:subClassOf o:ForProfitOrganization .  
5 | o:EducationInstitution rdf:subClassOf rdfs:Organization .  
6 | o:ForProfitOrganization rdf:subClassOf rdfs:Organization .
```

- rdfs:subClassOf = subset relation.
- Line 3 reads “every (instance of) o:University is a(n instance of) o:EducationInstitution.”, i.e., if $(x, \text{rdf:type}, \text{o:University})$ is true, then $(x, \text{rdf:type}, \text{o:EducationInstitution})$ must also be true.
- o:University, o:EducationInstitution, o:Company, o:ForProfitOrganization, and o:Organization are all classes.
- A class may have multiple subclasses and multiple superclasses.
- rdfs:subClassOf is **transitive**, hence can form a **class hierarchy**.

Properties

- RDF semantics says that every **predicate** of a triple:
 - semantically corresponds to a **binary relation**; and

Properties

- RDF semantics says that every **predicate** of a triple:
 - semantically corresponds to a **binary relation**; and
 - is called a **property** (recall the corresponding RDF entailment rule)

Properties

- RDF semantics says that every **predicate** of a triple:
 - semantically corresponds to a **binary relation**; and
 - is called a **property** (recall the corresponding RDF entailment rule)
- The IRI `rdf:Property` is in fact the class of all properties.

Properties

- RDF semantics says that every **predicate** of a triple:
 - semantically corresponds to a **binary relation**; and
 - is called a **property** (recall the corresponding RDF entailment rule)
- The IRI `rdf:Property` is in fact the class of all properties.
- Since it's a binary relation, a property can have domains and/or ranges, which are classes.

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:Course is an instance of rdfs:Class.

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:Course is an instance of rdfs:Class.
 - If $(x, \text{o:taughtBy}, y)$ is true, then x is an instance of o:Course.

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:Course is an instance of rdfs:Class.
 - If $(x, o:\text{taughtBy}, y)$ is true, then x is an instance of o:Course.
- Line 2 means/implies:

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:Course is an instance of rdfs:Class.
 - If $(x, o:\text{taughtBy}, y)$ is true, then x is an instance of o:Course.
- Line 2 means/implies:
 - o:taughtBy is an instance of rdf:Property.

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:Course is an instance of rdfs:Class.
 - If $(x, o:\text{taughtBy}, y)$ is true, then x is an instance of o:Course.
- Line 2 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:FacultyMember is an instance of rdfs:Class.

rdfs:domain and rdfs:range

```
1 | o:taughtBy rdfs:domain o:Course .  
2 | o:taughtBy rdfs:range o:FacultyMember .
```

- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:Course is an instance of rdfs:Class.
 - If $(x, o:\text{taughtBy}, y)$ is true, then x is an instance of o:Course.
- Line 2 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - o:FacultyMember is an instance of rdfs:Class.
 - If $(x, o:\text{taughtBy}, y)$ is true, then y is an instance of o:FacultyMember.

rdfs:subPropertyOf

```
1 | o:taughtBy rdfs:subPropertyOf o:involves .
```

- **rdfs:subPropertyOf** = subset between two binary relations.

rdfs:subPropertyOf

```
1 | o:taughtBy rdfs:subPropertyOf o:involves .
```

- rdfs:subPropertyOf = subset between two binary relations.
- Line 1 means/implies:

rdfs:subPropertyOf

```
1 | o:taughtBy rdfs:subPropertyOf o:involves .
```

- rdfs:subPropertyOf = subset between two binary relations.
- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.

rdfs:subPropertyOf

```
1 | o:taughtBy rdfs:subPropertyOf o:involves .
```

- rdfs:subPropertyOf = subset between two binary relations.
- Line 1 means/implies:
 - o:taughtBy is an instance of rdf:Property.
 - If $(x, o:\text{taughtBy}, y)$ is true, then $(x, o:\text{involves}, y)$ must also be true.

Example

```
1 :SemanticWeb o:taughtBy :Adila .
2 :Adila rdf:type o:Lecturer ; o:id "A234"; o:phone "+62-123-4567" .
3
4 o:Lecturer rdfs:subClassOf o:FacultyMember .
5 o:FacultyMember rdfs:subClassOf o:StaffMember .
6 o:taughtBy rdfs:domain o:Course ; rdfo:range o:FacultyMember ;
7             rdfs:subPropertyOf o:involves .
8 o:id rdfs:domain o:StaffMember ; rdfs:range rdfs:Literal .
9 o:phone rdfs:domain o:StaffMember ; rdfs:range rdfs:Literal .
```

What triples can we infer? Enumerate the elements of each class and property.

RDFS core classes

- rdfs:Class - class of all classes.
- rdf:Property - class of all properties.
- rdfs:Literal - class of all literal values.
- rdfs:Resource - class of all resources.
 - Resources are **everything**, including instances, classes, properties, literals, and datatypes
- rdfs:Datatype - class of all datatypes (for literals)
- rdf:Statement - class of all RDF statements

RDFS entailment

```
xxx ppp yyyy .  
ppp rdfs:domain zzz .
```

entails

```
xxx rdf:type zzz .
```

RDFS entailment

```
xxx ppp yyyy .  
ppp rdfs:domain zzz .
```

entails

```
xxx rdf:type zzz .
```

```
xxx ppp yyyy .  
ppp rdfs:range zzz .
```

entails

```
yyy rdf:type zzz .
```

RDFS entailment (cont.)

```
ppp rdfs:subPropertyOf qqq .  
qqq rdfs:subPropertyOf rrr .
```

entails

```
ppp rdfs:subPropertyOf rrr .
```

RDFS entailment (cont.)

```
ppp rdfs:subPropertyOf qqq .  
qqq rdfs:subPropertyOf rrr .
```

entails

```
ppp rdfs:subPropertyOf rrr .
```

```
xxx pppyyy .  
ppp rdfs:subPropertyOf qqq .
```

entails

```
xxx qqqyyy .
```

RDFS entailment (cont.)

```
ppp rdf:type rdf:Property .
```

entails

```
ppp rdfs:subPropertyOf ppp .
```

RDFS entailment (cont.)

```
ppp rdf:type rdf:Property .
```

entails

```
ppp rdfs:subPropertyOf ppp .
```

```
xxx pppyyy .
```

entails

```
xxx rdf:type rdfs:Resource .
```

```
yyy rdf:type rdfs:Resource .
```

RDFS entailment (cont.)

```
xxx rdf:type uuu .  
uuu rdfs:subClassOf vvv .
```

entails

```
xxx rdf:type vvv .
```

RDFS entailment (cont.)

```
xxx rdf:type uuu .  
uuu rdfs:subClassOf vvv .
```

entails

```
xxx rdf:type vvv .
```

```
uuu rdfs:subClassOf vvv .  
vvv rdfs:subClassOf www .
```

entails

```
uuu rdfs:subClassOf www .
```

RDFS entailment (cont.)

```
uuu rdf:type rdfs:Class .
```

entails

```
uuu rdfs:subClassOf  
      rdfs:Resource .  
uuu rdfs:subClassOf uuu .
```

RDFS entailment (cont.)

```
uuu rdf:type rdfs:Class .
```

entails

```
uuu rdfs:subClassOf  
      rdfs:Resource .
```

```
uuu rdfs:subClassOf uuu .
```

The following triples always hold for every datatype IRI ddd appearing in the graph:

```
ddd rdf:type rdfs:Datatype .
```

RDFS entailment (cont.)

```
uuu rdf:type rdfs:Class .
```

entails

```
uuu rdfs:subClassOf  
      rdfs:Resource .
```

```
uuu rdfs:subClassOf uuu .
```

The following triples always hold for every datatype IRI `ddd` appearing in the graph:

```
ddd rdf:type rdfs:Datatype .
```

Moreover,

```
ddd rdf:type rdfs:Datatype .
```

entails

```
ddd rdfs:subClassOf rdfs:Literal .
```

RDF/RDFS axiomatic triples

RDF/RDFS axiomatic triples are those that are set by the RDF/RDFS semantics to always be true.

```
rdf:type rdf:type rdf:Property .  
rdf:subject rdf:type rdf:Property .  
rdf:predicate rdf:type rdf:Property  
rdf:object rdf:type rdf:Property .  
rdf:first rdf:type rdf:Property .  
rdf:rest rdf:type rdf:Property .  
rdf:value rdf:type rdf:Property .  
rdf:nil rdf:type rdf>List .  
rdf:_1 rdf:type rdf:Property .  
rdf:_2 rdf:type rdf:Property .  
...  
...
```

```
rdf:type rdfs:domain rdfs:Resource .  
rdfs:domain rdfs:domain rdf:Property .  
rdfs:range rdfs:domain rdf:Property .  
rdfs:subPropertyOf rdfs:domain rdf:Property .  
rdfs:subClassOf rdfs:domain rdfs:Class .  
rdf:subject rdfs:domain rdf:Statement .  
rdf:predicate rdfs:domain rdf:Statement .  
rdf:object rdfs:domain rdf:Statement .  
rdfs:member rdfs:domain rdfs:Resource .  
rdf:first rdfs:domain rdf>List .  
rdf:rest rdfs:domain rdf>List .  
rdfs:seeAlso rdfs:domain rdfs:Resource .  
rdfs:isDefinedBy rdfs:domain rdfs:Resource .  
rdfs:comment rdfs:domain rdfs:Resource .  
rdfs:label rdfs:domain rdfs:Resource .  
rdf:value rdfs:domain rdfs:Resource .
```

RDF/RDFS axiomatic triples (cont.)

```
rdf:type rdfs:range rdfs:Class .  
rdfs:domain rdfs:range rdfs:Class .  
rdfs:range rdfs:range rdfs:Class .  
rdfs:subPropertyOf rdfs:range rdf:Property .  
rdfs:subClassOf rdfs:range rdfs:Class .  
rdf:subject rdfs:range rdfs:Resource .  
rdf:predicate rdfs:range rdfs:Resource .  
rdf:object rdfs:range rdfs:Resource .  
rdfs:member rdfs:range rdfs:Resource .  
rdf:first rdfs:range rdfs:Resource .  
rdf:rest rdfs:range rdf:List .  
rdfs:seeAlso rdfs:range rdfs:Resource .  
rdfs:isDefinedBy rdfs:range rdfs:Resource .  
rdfs:comment rdfs:range rdfs:Literal .  
rdfs:label rdfs:range rdfs:Literal .  
rdf:value rdfs:range rdfs:Resource .
```

```
rdf:Alt rdfs:subClassOf rdfs:Container .  
rdf:Bag rdfs:subClassOf rdfs:Container .  
rdf:Seq rdfs:subClassOf rdfs:Container .  
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .  
  
rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .  
  
rdfs:Datatype rdfs:subClassOf rdfs:Class .  
  
rdf:_1 rdf:type rdfs:ContainerMembershipProperty .  
rdf:_1 rdfs:domain rdfs:Resource .  
rdf:_1 rdfs:range rdfs:Resource .  
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .  
rdf:_2 rdfs:domain rdfs:Resource .  
rdf:_2 rdfs:range rdfs:Resource .  
...  
...
```

RDFS basic reasoning procedure

Given an RDF graph, RDFS reasoning computes all its entailed statements as follows:

1. Add all RDF/RDFS axiomatic triples except those containing the container membership property IRIs `rdf:_1`, `rdf:_2`, ...
2. For every container membership property IRI which occurs in the graph, add the axiomatic triples which contain that IRI.
3. Add triples to the graph according to the inference rules until no new triple can be added.

RDFS containers

- Vocabulary to describe containers (not constructing container like in programming languages).
 - Items in containers are enumerated
- No formal semantics: Intended just for human consumption
- Three types of containers: `rdf:Bag`, `rdf:Seq`, `rdf:Alt`.
- `rdf:_1`, `rdf:_2`, etc. are properties for enumerating. They are all subproperty of `rdfs:member` and all of them are instances of `rdfs:ContainerMembershipProperty`.

Example

```
[] rdf:type rdf:Bag ;  
rdf:_1 :itemA ;  
rdf:_2 :itemB .
```

```
[] rdf:type rdf:Seq ;  
rdf:_1 :itemC ;  
rdf:_2 :itemD .
```

```
[] rdf:type rdf:Alt ;  
rdf:_1 :choice1 ;  
rdf:_2 :choice2 .
```

“A bag with two items” Indicates (informally) that the container is intended to be unordered

“A sequence with two items” Indicates (informally) that the numerical ordering of the container membership properties is significant.

“An ‘Alternative’ container with two choices” Indicates (informally) that a typical processing is to select one of the members of the container.

RDFS collections

- Vocabulary to describe “list structure”.
- Unlike containers, collections:
 - can have branching structure, and
 - has an explicit terminator.
- One type of collection: `rdf>List`.
- Use the properties `rdf:first` and `rdf:rest`, as well as the resource `rdf:nil` to form the list structure (akin to LISP/Haskell)
- Can be hidden using parentheses syntax in Turtle.

Example

A list with three elements whose third element is a list with two elements.

```
:somelist s:content (:x :y (:z :w) ) .
```

is equivalent to

Example

A list with three elements whose third element is a list with two elements.

```
:somelist s:content (:x :y (:z :w) ) .
```

is equivalent to

```
:somelist :content _:genid1 .  
_:genid1 rdf:first :x ; rdf:rest _:genid2 .  
_:genid2 rdf:first :y ; rdf:rest _:genid3 .  
_:genid4 rdf:first :z ; rdf:rest _:genid5 .  
_:genid5 rdf:first :w ; rdf:rest rdf:nil .  
_:genid3 rdf:first _:genid4 ; rdf:rest rdf:nil .
```

RDFS container membership entailment

```
ppp rdf:type rdfs:ContainerMembershipProperty .
```

entails

```
ppp rdfs:subPropertyOf rdfs:member .
```

Other vocabulary terms

- None of these vocabulary terms have formal semantics!
- Utility properties
 - rdfs:label – human-readable label
 - rdfs:comment – for commenting
 - rdfs:seeAlso – pointing to other IRI that explains the resource
 - rdfs:isDefinedBy – subproperty of rdfs:seeAlso
- RDF reification (rarely used nowadays) – see the standards.
 - rdf:Statement
 - rdf:subject
 - rdf:predicate
 - rdf:object

Linked Data

Adila Krisnadhi





UNIVERSITAS
INDONESIA

Virtus, Prudentia, Industria

FACULTY OF
COMPUTER
SCIENCE



Hello!

I am Adila Krisnadhi

Faculty member at the Faculty of Computer Science, Universitas
Indonesia.

Co-director of Tokopedia-UI AI Center of Excellence
Ontology engineer and Semantic Web enthusiast



UNIVERSITAS
INDONESIA
Virtute, Prodigio, Intellectu

FACULTY OF
COMPUTER
SCIENCE

Credits

- ◊ Presentation template by [SlidesCarnival](#)
- ◊ Photographs by [Unsplash](#)
- ◊ Backgrounds by [SubtlePatterns](#)
- ◊ Figures are taken from Allemang, Hendler, Gandon, "Semantic Web for the Working Ontologist", 3rd Ed.



UNIVERSITAS
INDONESIA
Veritas Proficit In vita

FACULTY OF
COMPUTER
SCIENCE

Smart (Cognitive) Applications & Services

The Layer Cake: The Technology Components

Semantic Web of Linked Data

Document Types

RDF-NTriples, RDF-Turtle, RDF-XML, RDF-JSON, JSON-LD, others

Sentence Part Identifiers

HTTP IRIs & URIs

Abstract Language
RDF Subject->Predicate->Object Sentences

Rules
SWRL, SPIN, R2RML, SHACL

Query
SPARQL, SPARQL

Unifying Logic
First-Order Logic (FOL)

Dictionaries
(Ontologies)
RDF, RDFS, OWL,
SKOS, Schema.org

Proof

Trust

Transmission
Security
(Crypto)

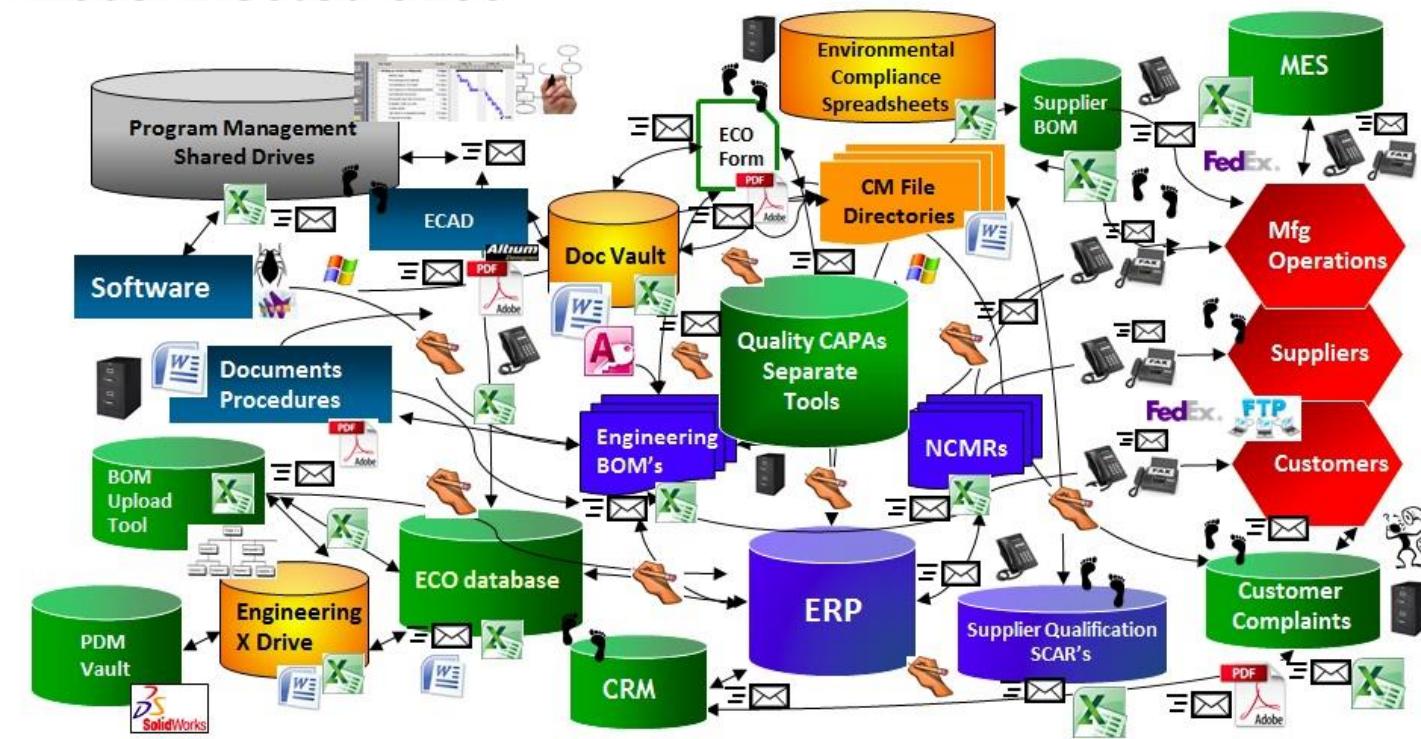


UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Traditional model: Data silos

<http://beyondplm.com/2014/07/22/plm-implementations-nuts-and-bolts-of-data-silos/>





UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Data on the Web vs. Web of Data

- ❖ Data on the Web = make data available on the Web, e.g., spreadsheets, web tables, PDFs.
- ❖ Web of data = data (from around the world) linked together so that it can be found, browsed, crawled, integrated, etc.
- ❖ Semantic Web → Web of data, not just data on the Web.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Open data vs. linked data



Open data

- Openly available (on the Web) under an open license.
- But not necessarily free to (re)use, e.g., due to proprietary file formats.
- “Open data and content can be freely used, modified, and shared by anyone for any purpose” – <http://opendefinition.org>
- Open data portals: data.gov, data.gov.uk, data.id, etc.



Linked data = Web of data

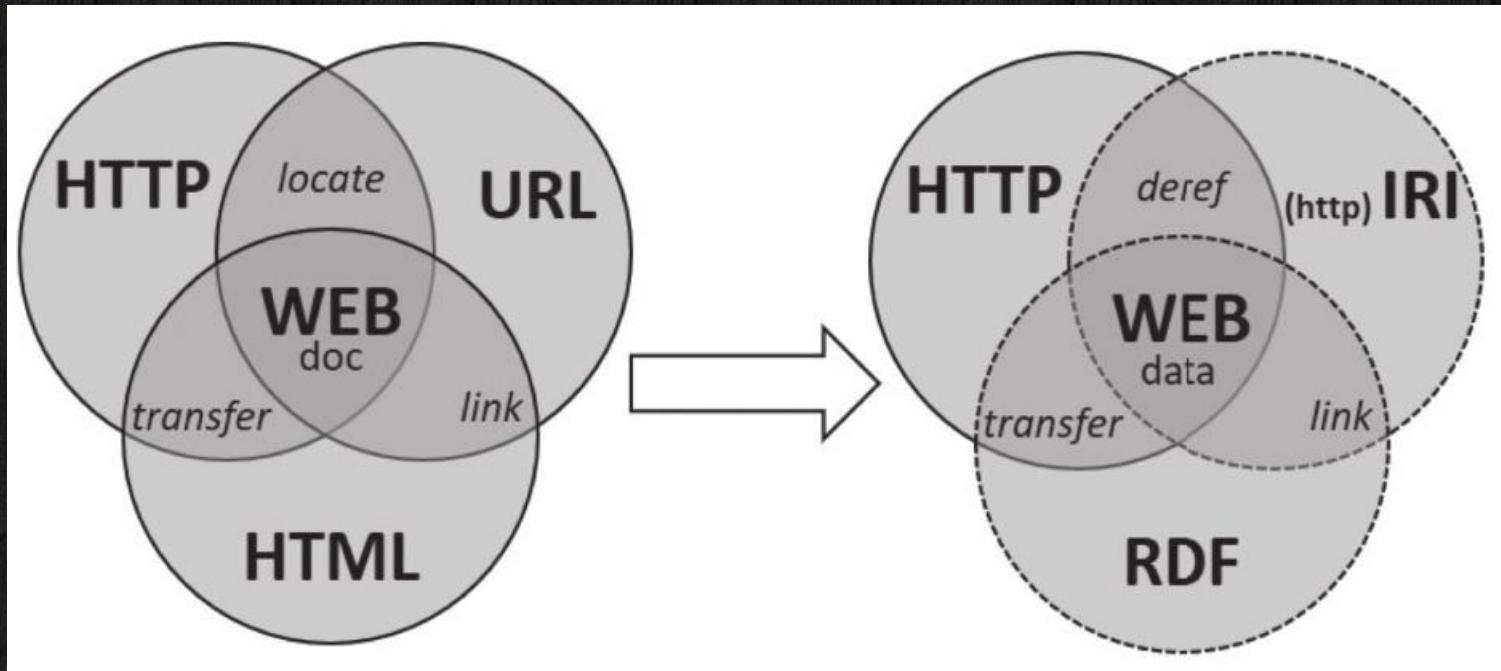
- Practical and simple Web of data
- Employs W3C Semantic Web standards focusing on simple semantics.
- Data may be closed.
- Linked Open Data = linked data that is also open data.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusa

FACULTY OF
COMPUTER
SCIENCE

How do we make (hypertext) Web allowing Web of data?





UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Web of Data: Content production

❖ **Embedded in web pages**

- Microformats
- Microdata
- RDFa
- JSON-LD

❖ **Standalone RDF data**

- RDF files
- APIs (backed by triple store, RDB, etc.)

Linked Data, Semantic Web



UNIVERSITAS
INDONESIA

Veritas. Pudicit. Intellit.

FACULTY OF
COMPUTER
SCIENCE

Microformats

- ❖ Structured data in embedded web pages.
- ❖ **Existing HTML tags** are used for information representation.
- ❖ Information represented with a set of properties and their values
- ❖ Formats tailored at specific need
- ❖ Enables more focused search, more accurate classification and visualization of search results.
- ❖ Can be transformed into RDF with GRDDL (XSLT)
- ❖ See examples at <https://microformats.org/wiki/microformats2>



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Microdata

- ❖ Extends HTML5 specification with primitives for information representation.
- ❖ Simple semantics (compared to Semantic Web standards)
- ❖ W3C Working Draft: <https://www.w3.org/TR/microdata/>
- ❖ Representative example vocabulary: schema.org
 - Schemas for structured data mark-up on the Web
 - Launched together in Juni 2011 by Google, Yahoo, Microsoft

[Full Hierarchy](#)

Schema.org is defined as two hierarchies: one for textual property values, and one for the things that they describe.

This is the main schema.org hierarchy: a collection of types (or "classes"), each of which has one or more parent types. Although a type may have more than one super-type, here we show each type in one branch of the tree only. There is also a parallel hierarchy for [data types](#).

- [Thing](#)
 - [Action](#)
 - [AchieveAction](#)
 - [LoseAction](#)
 - [TieAction](#)
 - [WinAction](#)
 - [AssessAction](#)
 - [ChooseAction](#)
 - [VoteAction](#)
 - [IgnoreAction](#)
 - [ReactAction](#)
 - [AgreeAction](#)
 - [DisagreeAction](#)
 - [DislikeAction](#)
 - [EndorseAction](#)
 - [LikeAction](#)

See examples at <https://schema.org/docs/documents.html>



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

RDFa

- ❖ W3C Recommendation to embed RDF on an XHTML page
 - Bridging human and data webs
 - Software can extract RDF graph for the machine.
 - See playground at <http://rdfa.info/>
 - RDFa distiller and parser:
<http://www.w3.org/2012/pyRdfa/>
- ❖ Schema.org support since RDFa 1.1
- ❖ Open Graph Protocol (OGP)
 - Major user of RDFa
 - Used in Facebook



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

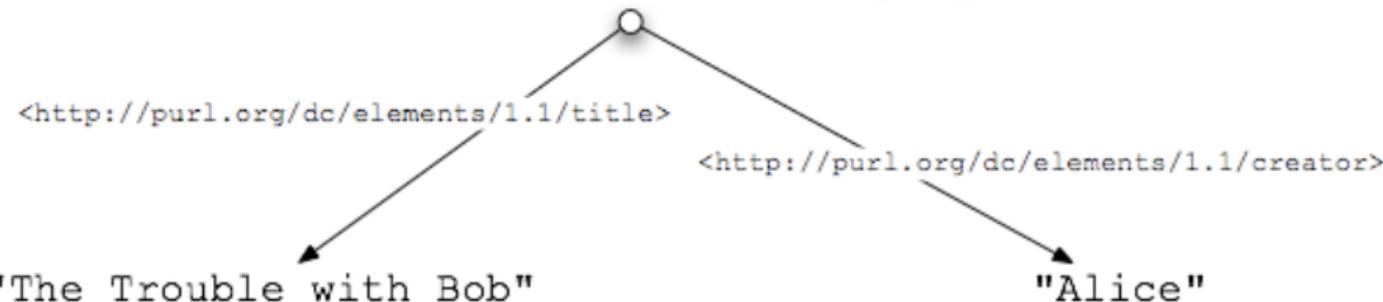
```
<div>
    <h2>The trouble with Bob</h2>
    <h3>Alice</h3>
    ...
</div>
```



```
<div xmlns:dc="http://purl.org/dc/elements/1.1/">
    <h2 property="dc:title">The trouble with Bob</h2>
    <h3 property="dc:creator">Alice</h3>
    ...
</div>
```

RDFa Example

`<http://example.com/alice/posts/trouble_with_bob>`



Literal Properties: RDFa lets Alice connect not just one URL to another—for example to connect her blog entry URL to the Creative Commons license URL—but also to connect one URL to a string such as "The Trouble with Bob". All arrows are labeled with the corresponding property name, which is also a URL.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

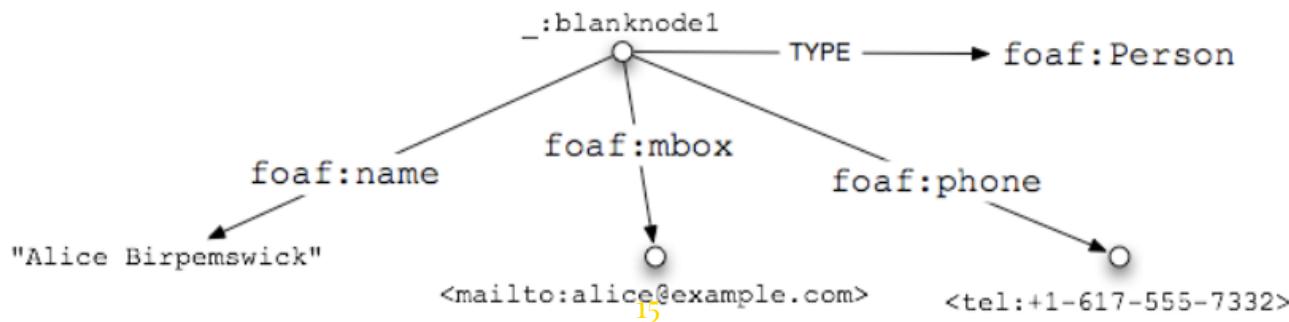
FACULTY OF
COMPUTER
SCIENCE

```
<div typeof="foaf:Person" xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <p property="foaf:name">
    Alice Birpemswick
  </p>

  <p>
    Email: <a rel="foaf:mbox" href="mailto:alice@example.com">alice@example.com</a>
  </p>

  <p>
    Phone: <a rel="foaf:phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
</div>
```

RDFA Example





UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

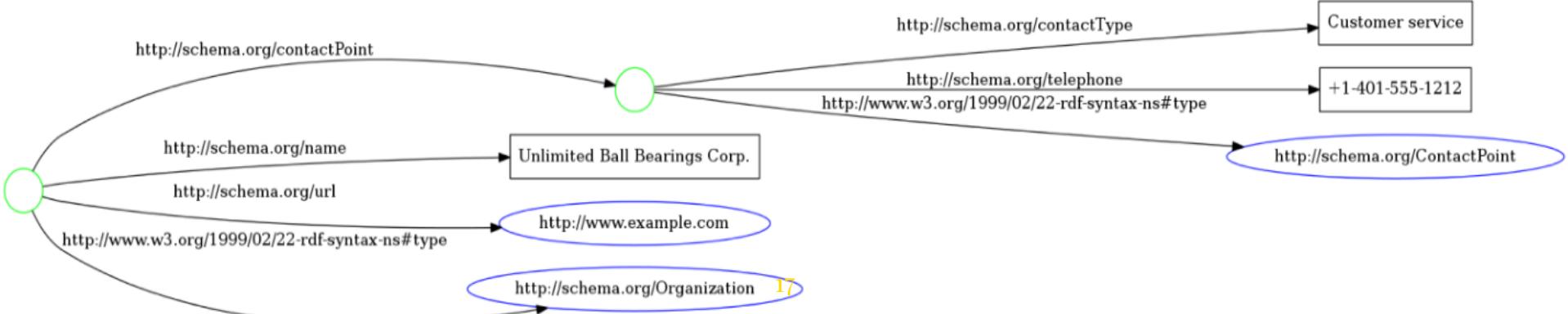
FACULTY OF
COMPUTER
SCIENCE

JSON-LD

- ❖ W3C Recommendation:
<https://www.w3.org/TR/json-ld11/>
- ❖ JSON-based RDF serialization format
- ❖ Can be directly embedded on an HTML page
- ❖ Schema.org supports
- ❖ Google recommends JSON-LD over RDFa and Microdata

JSON-LD example

```
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "Organization",
  "url": "http://www.example.com",
  "name": "Unlimited Ball Bearings Corp.",
  "contactPoint": {
    "@type": "ContactPoint",
    "telephone": "+1-401-555-1212",
    "contactType": "Customer service"
  }
}
</script>
```





UNIVERSITAS
INDONESIA
Virtus, Prudentia, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Linked Data: Main ideas

- ❖ Creation of (as many as possible) datasets in RDF
- ❖ Linking datasets together
 - Cross referencing data in other datasets
 - e.g., place "Indonesia" in GeoNames to president "Joko Widodo" in DBpedia
 - Identifying same concepts in different datasets
 - e.g., "Jakarta" in GeoNames and DBpedia
- ❖ Employing lightweight semantic technologies
- ❖ Linked Open Data (LOD) community



UNIVERSITAS
INDONESIA

Virtus, Prudentia, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Linked data principles

See <https://www.w3.org/DesignIssues/LinkedData.html> and <https://doi.org/10.2200/S00334ED1V01Y201102WBE001>

1. Use URIs to name everything.
2. Use HTTP URIs to allow people looking up those names.
3. When someone/agent looks up a URI, provide useful information using the standards (RDF and its syntaxes).
4. In the returned information, include links to HTTP URIs of other things so that agent can discover more things.



UNIVERSITAS

INDONESIA

Veritas. Pudicit. Inclusa

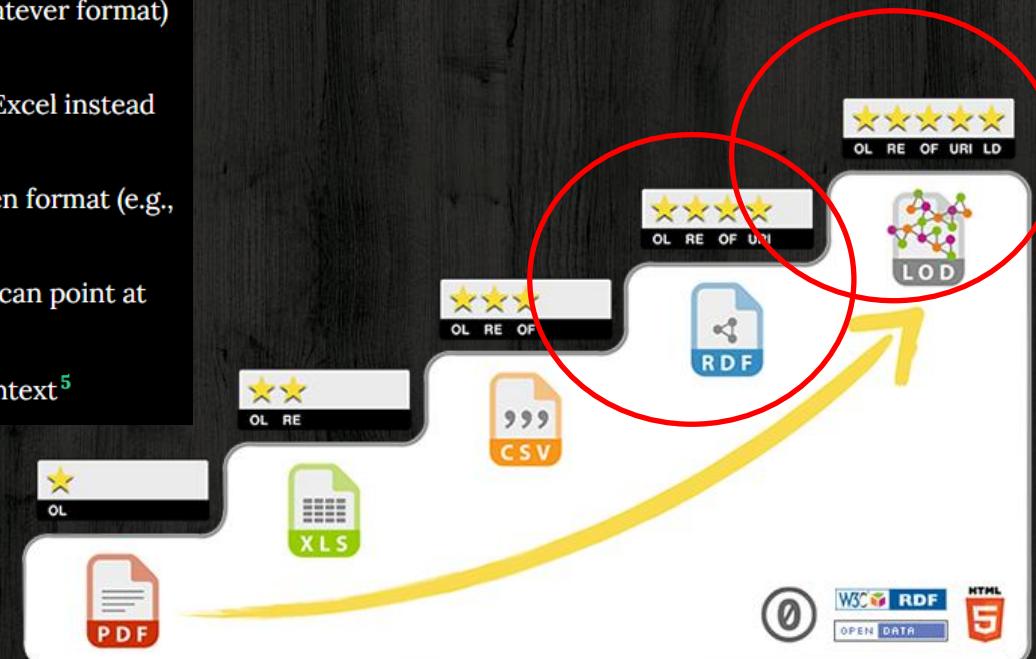
FACULTY OF

COMPUTER
SCIENCE

5-star rating in linked data publishing

- ★ make your stuff available on the Web (whatever format) under an open license¹
- ★★ make it available as structured data (e.g., Excel instead of image scan of a table)²
- ★★★ make it available in a non-proprietary open format (e.g., CSV instead of Excel)³
- ★★★★ use URIs to denote things, so that people can point at your stuff⁴
- ★★★★★ link your data to other data to provide context⁵

<https://5stardata.info/en/>



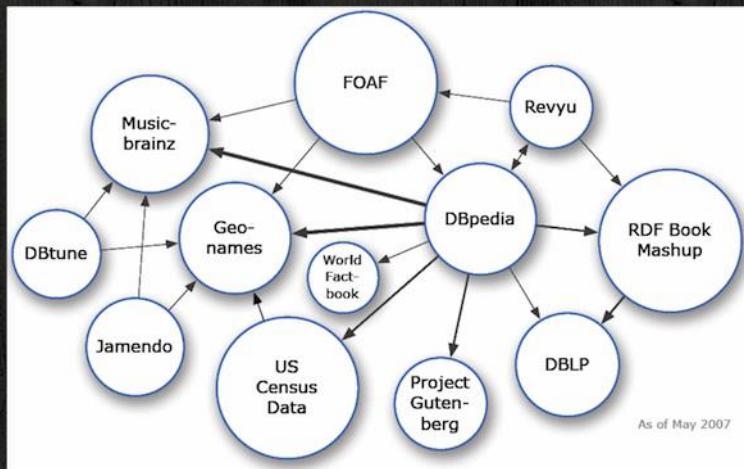


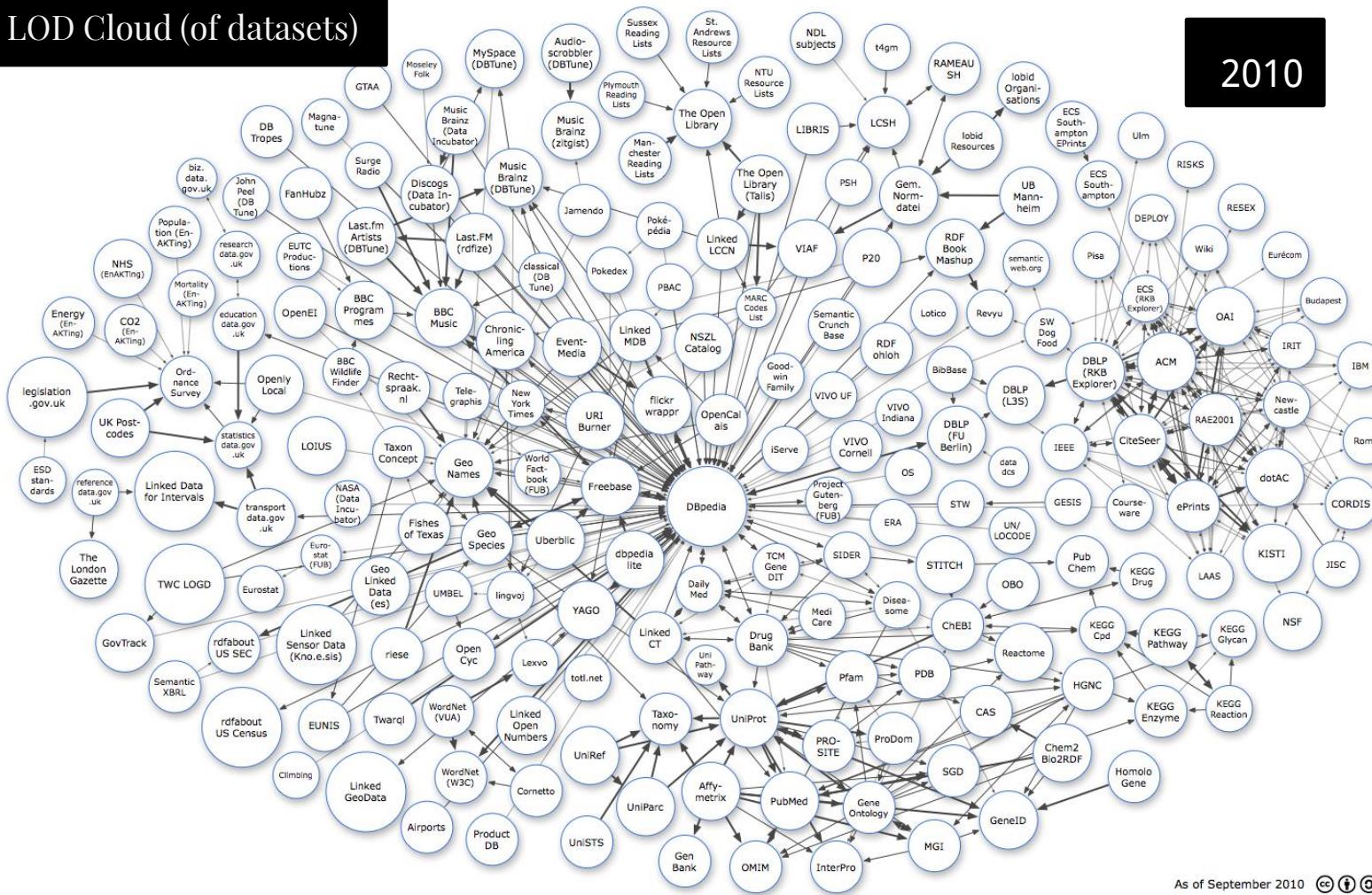
UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

LOD Cloud (of datasets)

2007







UNIVERSITAS
INDONESIA
Virtute, Prodigio, Intellectu

FACULTY OF
COMPUTER
SCIENCE

LOD Cloud

- ❖ 2022?
 - Too big to display here...
 - See <https://lod-cloud.net>



UNIVERSITAS
INDONESIA
Veritas. Pudicit. Inclusa

FACULTY OF
COMPUTER
SCIENCE

Knowledge graph in Google search

harry potter

Books

View 2+ more

Harry Potter and the Philosopher's Stone (1997)	Harry Potter and the Chamber of Secrets (1998)	Harry Potter and the Prisoner of Azkaban (1999)	Harry Potter and the Goblet of Fire (2000)	Harry Potter and the Order of the Phoenix (2003)
---	--	---	--	--

People also search for

View 5+ more

See results about

Fictional universe of Harry Potter
The Wizarding World is a fantasy media franchise and shared fictional universe ...

Feedback

People also ask

Is Snape Harry's father?
What is Harry Potter's full name?
Who is Harry Potter in the story?
Is Harry Potter a true story?

Feedback

Book character

Harry Potter (Book character)
Fictional universe: Fictional universe of Harry Potter
Played by: Daniel Radcliffe, Jacob Saunders, Charles Saunders

Fictional universe

Film series

Harry Potter (Film series)
Cast: Emma Watson, Rupert Grint, Tom Felton, Alan Rickman
Movies: Harry Potter and the Philosopher's Stone

Feedback

Books

Fictional
universe

Book
character

Film series



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF

COMPUTER
SCIENCE

Linked Data: How do we name things with URI?



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Naming Things with URIs to reach 4 stars

- ❖ Use HTTP URIs
 - Domain name system makes distributed URI minting easy
 - HTTP URIs can be used to access more information; so, avoid URNs, DOIs, etc.
- ❖ Need to avoid confusion between the objects themselves and the Web documents that describe them.
 - <http://www.cs.ui.ac.id> is a university or a Web document about it?



UNIVERSITAS
INDONESIA

Veritas. Pudicit. Intellit.

FACULTY OF

COMPUTER
SCIENCE

Cool URIs

- ❖ See: Tim Berners-Lee, "Cool URIs don't change", <http://www.w3.org/Provider/Style/URI> and <https://www.w3.org/TR/cooluris/>
- ❖ Cool URIs are **simple**
 - Prefer short, mnemonic URIs (not easily broken when copy-pasted and easier to remember)
- ❖ Cool URIs are **stable**
 - Once set-up to identify certain resources, URIs should live forever.
 - Keep implementation-specific out of URIs (e.g., .php, .asp)
- ❖ Cool URIs are **manageable**
 - Issue your URIs in a way that you can manage,
 - e.g., include current year (so can change URI schema each year without breaking older URIs)
 - Keep all 303 URIs on a dedicated subdomain, e.g., <http://id.example.com/alice> to ease later migration of the URI-handling system.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

URI dereferencing (3rd LD principle)

- ❖ **Content negotiation:** Returning the right information about a URI using HTTP
 - As specified by HTTP header request fields
 - E.g., Accept: text/plain
 - E.g., a web browser requests a HTML page.
- ❖ Server decides what to return.
- ❖ Two dereferencing strategies (to allow distinction between objects):
 - 303 URIs
 - Hash URIs



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

303 URIs

- ◊ Problem: real world objects/abstract concepts cannot be returned; only documents can.
- ◊ 303 URI strategy:
 - Server return a 303 See Other response code (HTTP redirect)
 - Client can find the related document by following the redirect, e.g., to obtain a HTML page or an RDF description.
- ◊ Example:
<http://data.finlex.fi/eli/sd/2008/521/luku/1/pykala/1/ajantasa/20160101>
is redirected to
<http://data.finlex.fi/eli/sd/2008/521/luku/1/pykala/1/ajantasa/20160101.html>
- ◊ Good for accessing parts of RDF descriptions (including the large ones).
 - Requires one extra HTTP request.
 - But only relevant data is transmitted and no further filtering is needed.



UNIVERSITAS
INDONESIA
Virtus, Prudentia, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Hash URIs

- ◆ 303 URI needs two requests when dereferencing
- ◆ Hash URI: use # and fragment identifier.
 - E.g., <http://example.org/vocab/course#Seminar>
- ◆ Procedure:
 - Client truncates URI at # (remove #Seminar)
 - Clients sends a GET request for <http://example.org/vocab/course>
 - Server returns either an RDF or HTML document.
 - The whole document is returned, not just info about #Seminar (because the fragment is not seen by the server).
- ◆ Good for small RDF descriptions (e.g., schemas)
 - Can save extra HTTP request
 - But the client must filter the response further.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

Linked data interfaces: How one interacts with it

- ❖ Human application interface.
 - Normal browsing/searching
- ❖ Reading RDF data of a URI, based on URI dereferencing
- ❖ Linked data browsing: Linked data browser interface based on URI dereferencing
 - Browsing based on RDF properties rendered in HTML (e.g., in DBpedia)
 - Using Linkd data browsers for the Web of data (e.g., <http://uriburner.com>)
- ❖ SPARQL endpoint: querying data in a standard way for mashups, etc.
- ❖ Download data as an RDF data dump.



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

DBpedia.org URI model

Use several URIs

- ❖ URI for the real-world object itself.
<http://dbpedia.org/resource/Jakarta>
- ❖ URI for a related information resource that describes the real-world object and has an HTML representation
<http://dbpedia.org/page/Jakarta>
- ❖ URIs for a related information resource that describes the real-world object and has an RDF representation
<http://dbpedia.org/data/Jakarta>
<http://dbpedia.org/data/Jakarta.ttl>
<http://dbpedia.org/data/Jakarta.json>

...



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Inclusio

FACULTY OF
COMPUTER
SCIENCE

In addition

- ❖ Wikipedia pages for Jakarta
 - <https://en.wikipedia.org/wiki/Jakarta>
 - https://id.wikipedia.org/wiki/Daerah_Khusus_Ibukota_Jakarta
- ...
- ❖ Jakarta as Wikidata item
 - <https://www.wikidata.org/wiki/Q3630>



UNIVERSITAS
INDONESIA
Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Data linking across datasets to reach 5 stars

Add as many (high quality) links as possible (manual or automated):

- ◆ **Relationship links:** pointers to additional information
 - E.g., persons to places where they lived.
- ◆ **Identity links:** pointers to similar resources in other datasets
- ◆ **Vocabulary/ontology links:** pointers to vocabulary terms in metadata models or ontologies in related datasets
 - E.g., use Dublin Core vocabulary to describe provenance information such as dates, authors, etc.
 - E.g., use keyword thesauri for subject descriptions.



UNIVERSITAS
INDONESIA
Virtute, Prodigio, Intellectu

FACULTY OF
COMPUTER
SCIENCE

Linked data publishing examples

- ❖ Wikidata, DBpedia, GeoNames, ...
- ❖ (Some) open data portals: data.gov.uk, data.gov
- ❖ ...
- ❖ See also W3C's Working Group Note: Data on the Web Best Practices Use Cases & Requirements,
<https://www.w3.org/TR/dwbp-ucr/>



UNIVERSITAS
INDONESIA

Veritas, Pudicit, Justitia

FACULTY OF
COMPUTER
SCIENCE

Fin.



Home > Courses > PROG. IK REGULAR > REG - Gasal 2022/2023 > [Reg] Semantic Web (Jejaring Semantik) Gasal 2022/... > Querying Data Graph > Worksheet/PR 3

Started on Tuesday, 27 September 2022, 9:50 PM

State Finished

Completed on Sunday, 2 October 2022, 12:07 AM

Time taken 4 days 2 hours

Grade Not yet graded

Information

Anda diberikan basis data mengenai musik

yang https://drive.google.com/file/d/1kcUu5uwVUz5jVB_pZb_9wMNjemg93jQK/view?usp=sharing

Tugas Anda adalah membuat beberapa kueri SPARQL untuk menjawab kebutuhan-kebutuhan informasi dari data sesuai soal yang diberikan.

Petunjuk penggerjaan:

- Untuk menyelesaikan soal ini, Anda dapat menggunakan triple store yang Anda sukai, atau jika Anda belum menginstal, silakan unduh BlazeGraph dengan mengikuti petunjuk pada tautan yang diberikan di laman kuliah ini di SCeLE. Petunjuk-petunjuk berikutnya berlaku jika Anda menggunakan BlazeGraph.
- Jalankan blazegraph.jar dengan Java Virtual Machine di command line/terminal sesuai petunjuk di dokumen QuickStart. Perhatikan alamat dan nomor port yang muncul di command line karena itu yang akan dipakai untuk mengakses blazegraph dari browser Anda.
- Upload/update data music.ttl yang Anda sudah unduh sebelumnya ke dalam BlazeGraph dengan mengakses tab Update di aplikasi BlazeGraph.
- Kueri SPARQL dapat dicoba-coba dengan mengakses tab Query di aplikasi BlazeGraph.

Question 1

Complete

Marked out of 5.00

Berikan nama semua kelas dan property di dalam data. Untuk setiap kelas, tampilkan berapa banyak instans unik dari kelas tersebut, lalu untuk setiap property, tampilkan berapa banyak pasangan/instans unik yang terhubung dengan property tersebut.

Catatan:

Suatu IRI P merupakan sebuah property, jika ia muncul sebagai predikat di setidaknya sebuah tripel di dalam graf.

Instans dari P adalah setiap pasangan unik x dan y sehingga tripel (x, P, y).

Suatu IRI C di dalam graf merupakan sebuah kelas jika C muncul dalam salah satu tripel berikut:

- (X, rdf:type, C).
- (A, rdfs:subClassOf, C)
- (C, rdfs:subClassOf, D)
- (P, rdfs:domain, C)
- (P, rdfs:range, C)

X merupakan instans dari kelas C apabila:

- (X, rdf:type, C) ada di graf, atau
- ada D sehingga (X, rdf:type, D) ada di graf dan D merupakan subclass dari C.

D merupakan subclass dari C apabila:

- C = D atau
- ada E sehingga (D, rdfs:subClassOf, E) ada di graf dan E merupakan subclass dari C.

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

```
SELECT * WHERE {
  {
    SELECT (?class as ?barang) ("class" as ?jenis) (COUNT(?instance) AS ?hitungBarang) WHERE {
      {
        SELECT DISTINCT ?class WHERE {
          {
            [ a ?class ]
          } UNION {
            [ rdfs:subClassOf ?class ]
          } UNION {
            ?class rdfs:subClassOf []
          } UNION {
            [ rdfs:domain ?class ]
          } UNION {
            [ rdfs:range ?class ]
          }
        }
      }
      # Simbol asterisk untuk mengecek 0 atau lebih rantai
      ?subclass rdfs:subClassOf* ?class .
      ?instance a ?subclass .
    }
    GROUP by ?class
  } UNION {
    SELECT (?prop as ?barang) ("property" as ?jenis) (COUNT(*) AS ?hitungBarang) WHERE {
      # Menggunakan blank node untuk menghitung banyaknya pasangan yang cocok
      [] ?prop []
    }
    GROUP BY ?prop
  }
}
```

Question 2

Complete

Marked out of 5.00

Cari nama semua artis beserta nama album-album yang pernah mereka rilis. Untuk setiap artis, berikan informasi apakah artis tersebut artis solo atau sebuah band. Lalu, untuk setiap album, kembalikan tanggal rilisnya jika ada. Urutkan hasilnya berdasarkan tanggal rilis albumnya.

Berdasarkan query berikut:

```
PREFIX : <http://domain.org/ns/>

SELECT (COUNT(DISTINCT *) as ?tanggalDouble) WHERE {
    ?album :date ?tanggal .
    ?album :date ?tanggal2 .
    FILTER (?tanggal < ?tanggal2)
}
ORDER BY ?tanggal
```

Terdapat beberapa album yang tanggal nya lebih dari satu, diasumsikan tanggal rilis merupakan tanggal yang paling kecil untuk setiap property `:date` pada sebuah album. Untungnya, setiap album memiliki setidaknya satu properti `:date`.

```
PREFIX : <http://domain.org/ns/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT * WHERE {
    ?album rdf:type :Album .
    OPTIONAL {
        ?album :name ?namaAlbum .
    }
    FILTER (!BOUND(?namaAlbum))
}
ORDER BY ?album
```

Berdasarkan query di atas, diketahui ada satu album bernama X dari Ed Sheeran yang tidak memiliki properti nama album. Sehingga ditulis query sebagai berikut.

```
PREFIX : <http://domain.org/ns/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT DISTINCT ?artis ?namaArtis ?jenisArtis ?album ?namaAlbum ?tanggal WHERE {
    ?jenisArtis rdfs:subClassOf :Artist .
    ?artis a ?jenisArtis .
    ?artis :name ?namaArtis
    OPTIONAL {
        ?album :artist ?artis .
        ?album a :Album .
        ?album :date ?tanggal .
        OPTIONAL { ?album :name ?namaAlbum }
    }
}
FILTER (
    NOT EXISTS
    {
        ?album :date ?tanggalLain
        FILTER (?tanggal > ?tanggalLain)
    } || !BOUND(?album)
)
ORDER BY ?tanggal
```

Question 3

Complete Marked out of 5.00

Cari semua nama band yang memiliki anggota yang juga merupakan artis solo. Kembalikan nama band tersebut beserta nama anggotanya yang menjadi artis solo.. Urutkan hasilnya berdasarkan nama band lalu berdasarkan nama artisnya.

```
PREFIX : <http://domain.org/ns/>
```

```
SELECT * WHERE {
    # Mencari band dan solo artis membernya
    ?band a :Band .
    ?band :member ?soloArtis .
    ?band :name ?namaBand .
    ?soloArtis a :SoloArtist .
    ?soloArtis :name ?namaSoloArtis .
}
ORDER BY ?namaBand ?namaSoloArtis
# Urutkan berdasarkan nama band, kemudian nama solo artis
```

Question 4

Complete

Marked out of 5.00

Cari semua judul album yang dirilis di dekade 1970an (tahun 1970 hingga 1979) yang dirilis oleh seorang artis solo dan diproduseri oleh artis itu sendiri (mungkin bersama produser lain). Selain judul album, tampilkan juga nama artis serta tanggal rilisnya.

PREFIX : <http://domain.org/ns/>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

```
SELECT ?namaArtis ?namaAlbum ?tanggal WHERE {  
    ?artis :name ?namaArtis .  
    ?artis a :SoloArtist .  
    ?album :producer ?artis .  
    ?album :artist ?artis .  
    ?album a :Album .  
    ?album :date ?tanggal .  
    OPTIONAL { ?album :name ?namaAlbum } .  
    FILTER (?tanggal >= "1970-01-01"^^xsd:date && ?tanggal < "1980-01-01"^^xsd:date)  
    FILTER NOT EXISTS {  
        ?album :date ?tanggalLain  
        FILTER (?tanggal > ?tanggalLain)  
    }  
}
```

Question 5

Complete Marked out of 5.00

Cari semua judul album yang dirilis di dekade 1980an (tahun 1980 hingga 1989) yang dirilis oleh sebuah band dan diproduseri oleh band itu sendiri atau oleh salah satu anggotanya (mungkin bersama produser lain). Selain judul album, tampilkan juga nama band, tanggal rilis, beserta jumlah track yang ada di dalam album tersebut.

```
PREFIX : <http://domain.org/ns/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

# Aggregation dengan group album artis tanggal
SELECT DISTINCT ?album ?artis ?tanggal (COUNT(DISTINCT ?lagu) as ?laguCount) WHERE {
    ?artis :name ?namaArtis .
    ?artis a :Band .
    ?album :artist ?artis .
    ?album a :Album .
    ?album :producer ?produser .
    ?album :date ?tanggal .
    # Mengecek semua lagu pada album ini
    ?album :track ?lagu .
    # Left join dengan nama album
    OPTIONAL { ?album :name ?namaAlbum } .
    FILTER (?tanggal >= "1980-01-01"^^xsd:date && ?tanggal < "1990-01-01"^^xsd:date)
    FILTER NOT EXISTS {
        ?album :date ?tanggallain # Mengecek tanggal terkecil
        FILTER (?tanggal > ?tanggallain)
    }
    # Mengecek produsernya merupakan artis (band) atau membernya langsung
    FILTER (?produser = ?artis || 
            EXISTS { ?artis :member ?produser . })
}
GROUP BY ?album ?artis ?tanggal
```

Question 6

Complete

Marked out of 5.00

Cari 10 nama penulis lagu dengan jumlah lagu terbanyak (yang ditulisnya). Untuk setiap penulis lagu tersebut, tampilkan juga total berapa album yang berisi lagu-lagu yang ditulisnya. Urutkan hasilnya berdasarkan jumlah lagu.

PREFIX : <http://domain.org/ns/>

```
SELECT ?penulis ?banyakTulisan ?countAlbum {  
  # Mencari penulis dengan jumlah lagu terbanyak  
  SELECT ?penulis (COUNT(?lagu) as ?banyakTulisan) WHERE {  
    ?lagu a :Song .  
    ?lagu :writer ?penulis  
  }  
  GROUP BY ?penulis  
  ORDER BY DESC(?banyakTulisan) # Limit ke 10 teratas saja setelah diurutkan  
  LIMIT 10  
}  
{  
  # Join dengan kontribusinya terhadap album  
  SELECT (?penulis2 as ?penulis) (COUNT(?album) as ?countAlbum) WHERE {  
    SELECT DISTINCT ?penulis2 ?album WHERE {  
      ?lagulain :writer ?penulis2 .  
      ?album :track ?lagulain  
    }  
    ORDER BY ?penulis2  
  }  
  GROUP BY ?penulis2  
}  
}  
ORDER BY DESC(?banyakTulisan)
```

Question 7

Complete

Marked out of 5.00

Cari nama semua band yang beranggotakan minimal 5 orang. Tampilkan nama band beserta jumlah anggotanya.

PREFIX : <http://domain.org/ns/>

```
SELECT * WHERE {
{
# Mencari jumlah anggota untuk semuanya
SELECT ?band ?namaBand (COUNT(?anggota) AS ?anggotaCount) WHERE {
?band a :Band .
?band :member ?anggota .
?band :name ?namaBand .
}
GROUP BY ?band ?namaBand
}
# Filter untuk anggotanya >= 5 saja
FILTER (?anggotaCount >= 5)
}
```

Question 8

Complete

Marked out of 5.00

Cari semua track/lagu terpanjang maupun terpendek dari setiap album. Tampilkan judul album, judul track, nama artis (band/artis solo), panjang track, beserta nama pencipta lagunya (ditulis dalam satu baris menggunakan GROUP CONCAT dengan separator koma jika suatu lagu memiliki beberapa pencipta). Urutkan berdasarkan panjang track.

PREFIX : <http://domain.org/ns/>

```
SELECT ?judulAlbum ?judulLagu ?namaArtis ?panjangLagu ?paraPenulis WHERE {
    ?album :artist ?artis .
    ?album :name ?judulAlbum .
    ?artis :name ?namaArtis .
    {
        # Mencari semua data lagu dengan panjang dan concat dari penulisnya
        SELECT ?album ?lagu ?judulLagu ?panjangLagu (GROUP_CONCAT(
            ?namaPenulis;SEPARATOR=", ") AS ?
        paraPenulis) WHERE {
            ?album :track ?lagu .
            ?lagu :writer ?penulis .
            ?penulis :name ?namaPenulis .
            ?lagu :length ?panjangLagu .
            ?lagu :name ?judulLagu
        }
        GROUP BY ?album ?lagu ?judulLagu ?panjangLagu
    }
    # Melakukan filter untuk mengecek apakah tidak ada lagu yang lebih panjang dari p
    ada dia
    FILTER ((NOT EXISTS
        {
            ?album :track ?laguLain .
            ?laguLain :length ?panjangLaguLain .
            ?laguLain :name ?judulLaguLain .
            FILTER (
                ?panjangLaguLain > ?panjangLagu ||
                (?panjangLaguLain = ?panjangLagu && ?judulLaguLain > ?judulLagu )
            )
        }
    ) ||
    # Melakukan filter untuk mengecek apakah tidak ada lagu yang lebih pend
    ek dari pada dia
    (NOT EXISTS
        {
            ?album :track ?laguLain .
            ?laguLain :length ?panjangLaguLain .
            ?laguLain :name ?judulLaguLain .
            FILTER (
                ?panjangLaguLain < ?panjangLagu ||
                (?panjangLaguLain = ?panjangLagu && ?judulLaguLain < ?judulLagu )
            )
        }
    )
)
}

ORDER BY ?panjangLagu # Urut berdasarkan panjang lagu
```