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

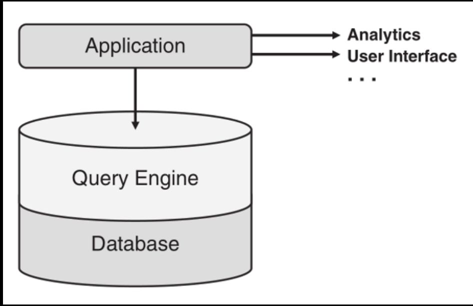
RDF application architecture: Overview

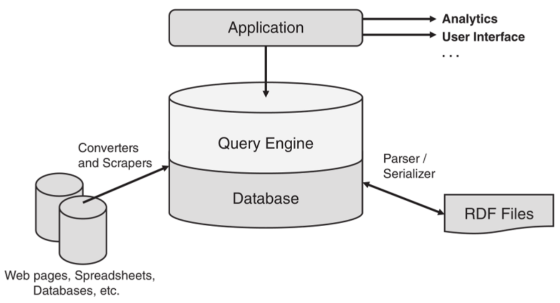
- Where do RDF data come from?
  - Manually created.
  - RDF files provided by others $\Rightarrow$ we need RDF parsers and RDF serializers for reading and writing RDF files.
  - Non-RDF files (spreadsheets, web pages, etc.) $\Rightarrow$ we need converters and scrapers.
  - Relational databases $\Rightarrow$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

RDF application architecture: Overview (cont.)

- 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) $\Rightarrow$ 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

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.

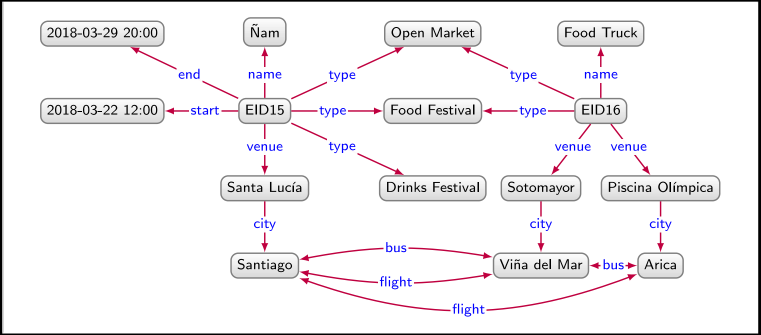
Basic graph patterns

- 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



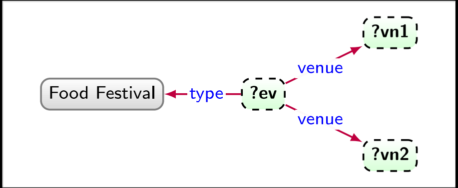
We wish to query food festivals. The answer should include two, possibly the same, venues of the events.

Example (cont.)

Then the corresponding BGP is:

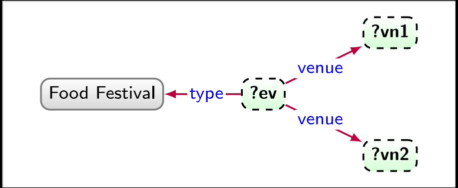
Example (cont.)

Then the corresponding BGP is:



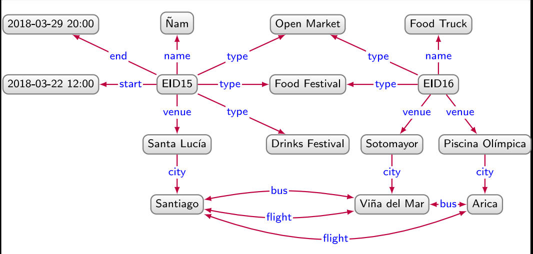
Example (cont.)

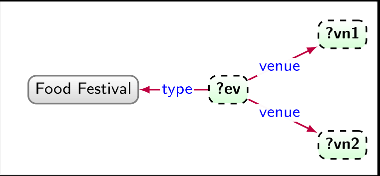
Then the corresponding BGP is:



Yielding answers ...

Example (cont.)

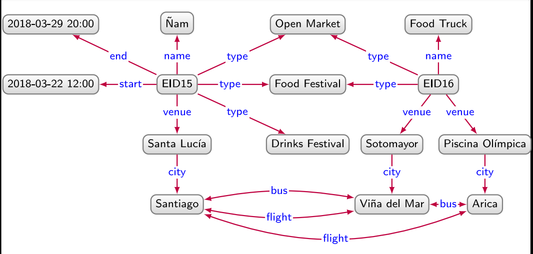


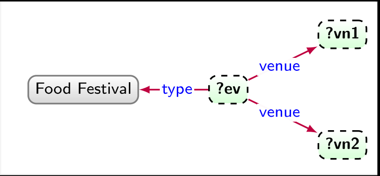


Yielding answers (with homorphism-based semantics):

?ev	?vn1	?vn2

Example (cont.)

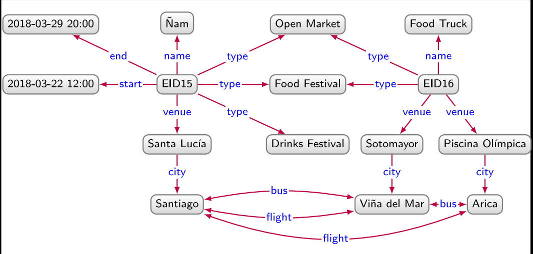


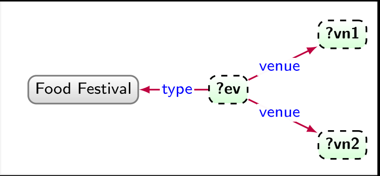


Yielding answers (with homorphism-based semantics):

?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor

Example (cont.)

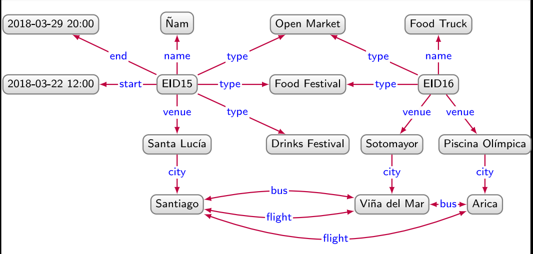


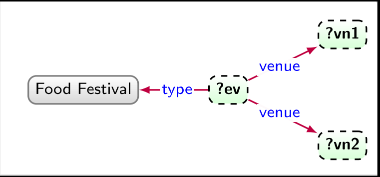


Yielding answers (with homorphism-based semantics):

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

Example (cont.)

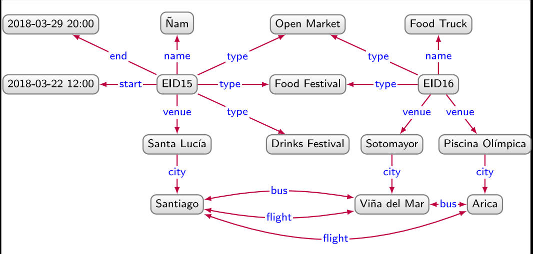


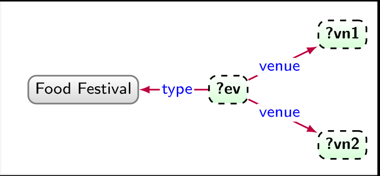


Yielding answers (with homorphism-based semantics):

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

Example (cont.)

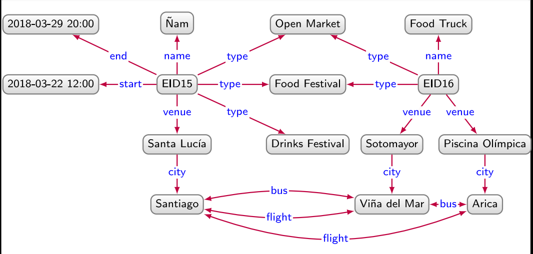


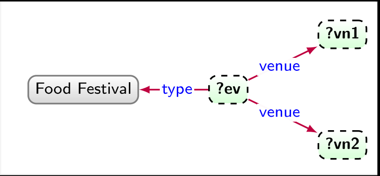


Yielding answers (with homorphism-based semantics):

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

Example (cont.)

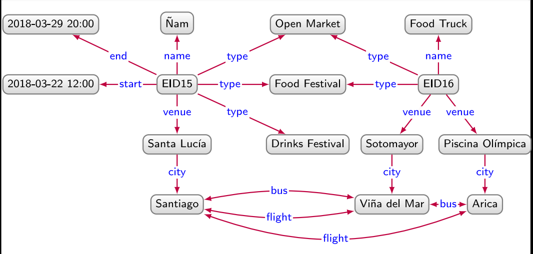


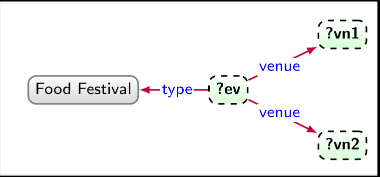


Yielding answers (with homorphism-based semantics):

?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 homorphism-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

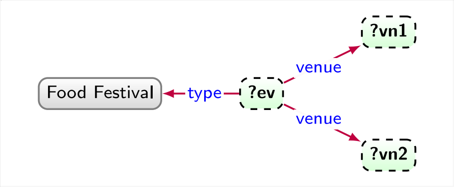
- Standard: `https://www.w3.org/TR/sparql11-overview/`
- Query language (syntax and semantics)
- RDF graph update through SPARQL
- Graph Store HTTP Protocol: HTTP operations for managing a graph collection
- Entailment regimes: query results with (additional) inferences
- Service description: methods 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

- $\mathbf{VAR}$: set of variables
- $\mathbf{IRI}$: set of IRIs
- $\mathbf{BN}$: set of blank nodes
- $\mathbf{LIT}$: set of literals
- $\mathbf{CON}$: set of constants $\rightarrow \mathbf{CON} = \mathbf{IRI} \cup \mathbf{BN} \cup \mathbf{LIT}$
- $\mathbf{VAR}$, $\mathbf{IRI}$, $\mathbf{BN}$, and $\mathbf{LIT}$ are pairwise disjoint.

- A triple pattern is a triple (s, p, o) where
  - subject $s \in \mathbf{VAR} \cup \mathbf{IRI} \cup \mathbf{BN}$ – (variables, IRI, or blank nodes)
  - predicate $p \in \mathbf{VAR} \cup \mathbf{IRI}$ – (variables or IRIs)
  - object $o \in \mathbf{VAR} \cup \mathbf{IRI} \cup \mathbf{BN} \cup \mathbf{LIT}$ – (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:

{(?ev, type, Food Festival),
(?ev, venue, ?vn1),
(?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, $\mathbf{VAR}(Q)$ denotes the set of all variables appearing in Q.
- For a partial mapping $\mu : \mathbf{VAR} \to \mathbf{CON}$ (from variables to constants)
  - $\mathsf{dom}(\mu)$ denotes the domain of μ, i.e., the set of variables for which μ is defined
  - given a variable $v \in \mathsf{dom}(\mu)$, $\mu(v)$ 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 $v \in \mathbf{VAR}(Q) \cap \mathsf{dom}(\mu)$ is replaced in Q by the constant $\mu(v)$.
- $\mathbf{VAR}(Q)$ may contain variables NOT in $\mathsf{dom}(\mu)$ and vice versa.
  - If $\mathbf{VAR}(Q) \subseteq \mathsf{dom}(\mu)$ holds, $\mu(Q)$ is in fact an RDF data graph (because it does not contain any variable).
  - If $\mathsf{dom}(\mu) \subsetneq \mathbf{VAR}(Q)$, $\mu(Q)$ remains a BGP (because it still contains variables).

BGP evaluation (cont.)

- A partial mapping $\mu : \mathbf{VAR} \to \mathbf{CON}$ 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:

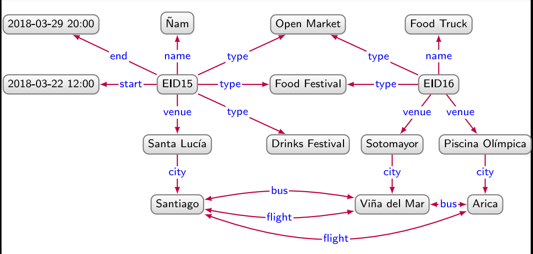
$$Q(G) = \{\mu \mid \mu(Q) \subseteq G \text{ and } \mathrm{dom}(\mu) \subseteq \mathbf{VAR}(Q)\}$$

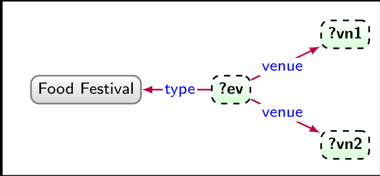
It's a multiset: duplicates of solution mappings are allowed and solution mappings are unordered.

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, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

1. $\mu = \{$?ev $\mapsto$ EID16, ?vn1 $\mapsto$ Piscina Olímpica, ?vn2 $\mapsto$ Sotomayor$\}$

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$
BGP $Q = \{$(?ev, type Food Festival), (?ev, venue, ?vn1), (?ev, venue, ?vn2)$\}$

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$
BGP $Q = \{$(?ev, type Food Festival), (?ev, venue, ?vn1), (?ev, venue, ?vn2)$\}$

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$
BGP $Q = \{$(?ev, type Food Festival), (?ev, venue, ?vn1), (?ev, venue, ?vn2)$\}$

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

4. $\mu = \{$?ev $\mapsto$ EID16, ?vn1 $\mapsto$ Sotomayor, ?vn2 $\mapsto$ Sotomayor$\}$
   $\mu(Q) = \{$(EID16, type, Food Festival), (EID16, venue, Sotomayor)$\}$

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

4. $\mu = \{$?ev $\mapsto$ EID16, ?vn1 $\mapsto$ Sotomayor, ?vn2 $\mapsto$ Sotomayor$\}$
   $\mu(Q) = \{$(EID16, type, Food Festival), (EID16, venue, Sotomayor)$\}$
   Is $\mu(Q) \subseteq G$?

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$
BGP $Q = \{$(?ev, type Food Festival), (?ev, venue, ?vn1), (?ev, venue, ?vn2)$\}$

4. $\mu = \{$?ev $\mapsto$ EID16, ?vn1 $\mapsto$ Sotomayor, ?vn2 $\mapsto$ Sotomayor$\}$
   $\mu(Q) = \{$(EID16, type, Food Festival), (EID16, venue, Sotomayor)$\}$
   Is $\mu(Q) \subseteq G$? Yes.

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

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

Example (cont.)

Data graph $G = \{$(EID,name, Ñam), $\ldots$, (EID15, type, Food Festival),
(EID, venue, Santa Lucía), $\ldots$,
(EID16, type, Food Festival), $\ldots$, (EID16, venue, Sotomayor),
(EID16, venue, Piscina Olímpica), $\ldots\}$

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

4. $\mu = \{$?ev $\mapsto$ EID16, ?vn1 $\mapsto$ Sotomayor, ?vn2 $\mapsto$ Sotomayor$\}$
   $\mu(Q) = \{$(EID16, type, Food Festival), (EID16, venue, Sotomayor)$\}$
   Is $\mu(Q) \subseteq G$? Yes.
5. $\mu = \{$?ev $\mapsto$ EID15, ?vn1 $\mapsto$ Santa Lucía, ?vn2 $\mapsto$ Santa Lucía$\}$
   $\mu(Q) = \{$(EID15, type, Food Festival), (EID15, venue, Santa Lucía)$\}$
   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:
  - $\mathbf{VAR}(Q) = \emptyset$, i.e., the query contains no variable;
  - the projection variables of the query is disjoint with the $\mathbf{VAR}(Q)$ (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:
  - $\mathbf{VAR}(Q) = \emptyset$, i.e., the query contains no variable;
  - the projection variables of the query is disjoint with the $\mathbf{VAR}(Q)$ (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 $\mu : \mathbf{VAR} \cup \mathbf{BN} \to \mathbf{CON}$ 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 $\text{domvar}(\mu) = \text{dom}(\mu) \cap \mathbf{VAR}$ – the set of variables for which μ is defined.
  - We define $\mu_{\text{var}} = \{v \mapsto \mu(v) \mid v \in \mathbf{VAR}\}$ – 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:

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

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:
$\{ \{?s \mapsto \text{ex:ex1}\}, \{?s \mapsto \text{ex:ex2}\} \}$

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:
$\{ \{?s \mapsto \text{ex:ex3}\} \}$

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:
$\{\ \{?s \mapsto \text{ex:ex4}\}, \{?s \mapsto \text{ex:ex5}\}\ \}$

Outline

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

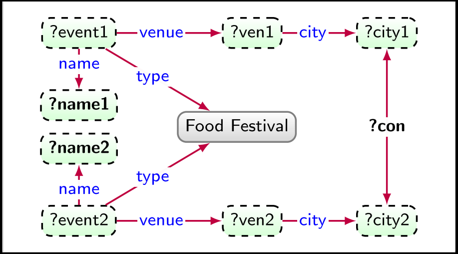
- Projection is realized in SPARQL via the SELECT output form.
- SELECT *varlist* returns a sequence of solution mappings restricted to the given variables in *varlist*.
    - If a variable in *varlist* does not occur in the graph pattern, it will be returned unbound.
    - If **all** variables in SELECT 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: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 .
}
```

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

FACULTY OF COMPUTER SCIENCE
UNIVERSITAS INDONESIA

```
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 $\{\ldots\}$.
    - 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.
  - P1 OPTIONAL { P2 } means: get all solution mappings for P1, and then optionally join with solution mappings of P2 if any.
  - If a solution mapping for P1 cannot be joined with any solution mapping for P2, then the solution mapping for P1 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: `!` $\rightsquigarrow$ `!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( expr1, expr2, expr3 )`	returns EBV of `expr2` if `expr1` is **true**, otherwise returns EBV of `expr3`
`EXISTS \{ pattern \}`	**true** if `pattern` matches; **false** otherwise
`NOT EXISTS \{ pattern \}`	**false** if `pattern` matches; **true** otherwise
`sameTerm( term1, term2 )`	**true** if `term1` and `term2` are the same; **false** otherwise.
	more general than = operator
`term IN ( expr1, ... )`	**true** if `term` can be found in the list on the right hand side
`term NOT IN ( expr1, ...)`	**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 `lit`.
`STR( res )`	returns the codepoint/string representation of the IRI `res`
`LANG( lit )`	returns the language tag of the literal `lit`, if any; returns `""` otherwise
`DATATYPE( lit )`	returns the datatype of `lit`
`IRI( lit )`, `IRI( res )`	returns an IRI from the literal `lit` or an IRI `res` `lit` 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 `list` and datatype `res`
`STRLANG( lit, ltag )`	creates a language-tagged literal with lexical form `list` and language tag `ltag`
`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 P1 MINUS P2, P2 can only remove matches in P1 if P1 and P2 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:cofee1	

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:cofee1	

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

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. `^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|...|irin)` – an IRI not one of `iri1`,..., `irin`.
9. `!(^iri1|...|^irin)` – an IRI not one of reverse of `iri1`, ..., `irin`. 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

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