

NDBI050: Query Languages II

<http://www.ksi.mff.cuni.cz/~svoboda/courses/232-NDBI050/>

Lecture

SPARQL

Martin Svoboda

martin.svoboda@matfyz.cuni.cz

30. 4. 2024

Charles University, Faculty of Mathematics and Physics

Lecture Outline

RDF stores

- Introduction
- Linked Data

RDF data format

- Framework, notations

SPARQL query language

- Graph patterns
- Filter constraints
- Solution modifiers
- Aggregation
- Query forms

RDF Stores

Data model

- **RDF triples**
 - Components: **subject**, **predicate**, and **object**
 - Each triple represents a **statement** about a real-world entity
- Triples can be viewed as **graphs**
 - **Vertices** for subjects and objects
 - **Edges** directly correspond to individual statements

Query language

- **SPARQL**: *SPARQL Protocol and RDF Query Language*

Representatives

- Apache **Jena**, **rdf4j** (Sesame), Algebraix
- *Multi-model*: **MarkLogic**, OpenLink **Virtuoso**

Linked Data

Linked Data

- Method of **publishing structured and interlinked data** in a way that allows for an **automated processing by programs** rather than browsing by human readers

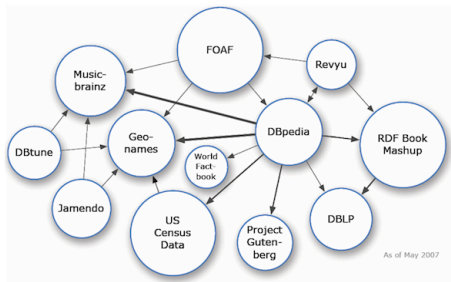
Principles of Linked Open Data

- **Identify resources** using URIs or even better using **URLs**
- **Publish data** about resources in standard formats via **HTTP**
- Mutually **interlink resources** to form Web of Data
- Release the data under an **open licence**

Linked Open Data Cloud

May 2007

- 12 datasets



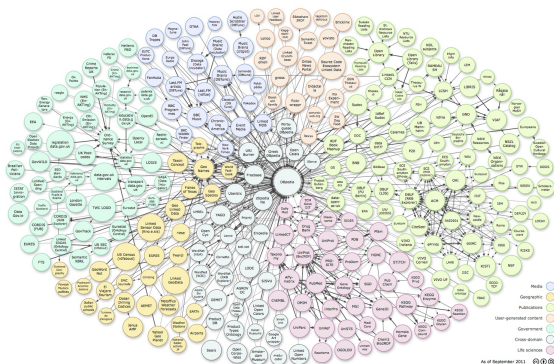
October 2007

- 25 datasets, 2 billion triples, 2 million links

Linked Open Data Cloud

September 2011

- 295 datasets, 31 billion triples, 504 million links

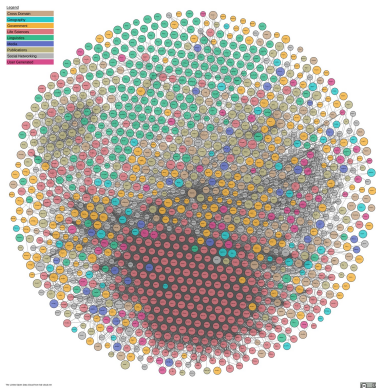


Source: <http://lod-cloud.net/>

Linked Open Data Cloud

March 2019

- 1239 datasets, 16 thousand dataset links



Source: <http://lod-cloud.net/>

RDF

Resource Description Framework

Introduction

RDF = *Resource Description Framework*

- Language for **representing information about resources** in the World Wide Web
 - + a family of related technologies, languages, specifications, ...
 - Used in the context of the **Semantic Web, Linked Data, ...**
- Developed by **W3C**
- Started in 1997
- Versions: 1.0 and 1.1
- W3C recommendations
 - <https://www.w3.org/TR/rdf11-concepts/>
 - Concepts and Abstract Syntax
 - <https://www.w3.org/TR/rdf11-mt/>
 - Semantics

Statements

Resource = any real-world entity

- **Referents** = **resources identified by IRIs**
 - Physical things, documents, abstract concepts, ...
 - E.g.: <http://db.cz/movies/medvidek>
- **Values** = **resources for literals**
 - Numbers, strings, ...
 - E.g.: [Medvidek](#)

Statement about resources = one RDF **triple**

- Three components: **subject**, **predicate**, and **object**

Examples

```
<http://db.cz/movies/medvidek>  
<http://db.cz/terms#title>  
"Medvidek" .
```

```
<http://db.cz/movies/medvidek>  
<http://db.cz/terms#actor>  
<http://db.cz/actors/trojan> .
```

Statements

Triple components

- **Subject**
 - Describes a resource the given statement is about
 - IRI or blank node identifier
- **Predicate**
 - Describes the property or characteristic of the subject
 - IRI
- **Object**
 - Describes the value of that property
 - IRI or blank node identifier or literal

Although triples are inspired by natural languages, they have nothing to do with NLP (Natural Language Processing)

Example

```
<http://db.cz/movies/medvidek>
  <http://db.cz/terms#title> "Medvídek" .

<http://db.cz/movies/medvidek>
  <http://db.cz/terms#actor> <http://db.cz/actors/machacek> .
<http://db.cz/movies/medvidek>
  <http://db.cz/terms#actor> <http://db.cz/actors/trojan> .

<http://db.cz/movies/medvidek>
  <http://db.cz/terms#year> "2007" .

<http://db.cz/movies/medvidek>
  <http://db.cz/terms#director> _:n18 .
_:n18
  <http://db.cz/terms#firstname> "Jan" .
_:n18
  <http://db.cz/terms#lastname> "Hřebejk" .
```

Identifiers and Literals

IRI = *Internationalized Resource Identifier*

- Absolute (not relative) IRIs with optional fragment identifiers
- RFC 3987
- Unicode characters
- Examples
 - `http://db.cz/movies/medvidek`
 - `http://db.cz/terms#actor`
 - `mailto:svoboda@ksi.mff.cuni.cz`
 - `urn:issn:0167-6423`
- **URLs are often used** in practice → information about given resources are then intended to be published / retrieved via standard HTTP

Identifiers and Literals

Literals

- Plain values
 - "Medvídek", "2007"
- Typed values
 - "2007"^^<http://www.w3.org/2001/XMLSchema#int>
 - XML Schema simple data types are usually used
 - xsd:string, xsd:integer, xsd:date, ...
- Strings with language tags
 - "Medvídek"@cs
- *Types and language tags cannot be mutually combined*

Identifiers and Literals

Blank node identifiers

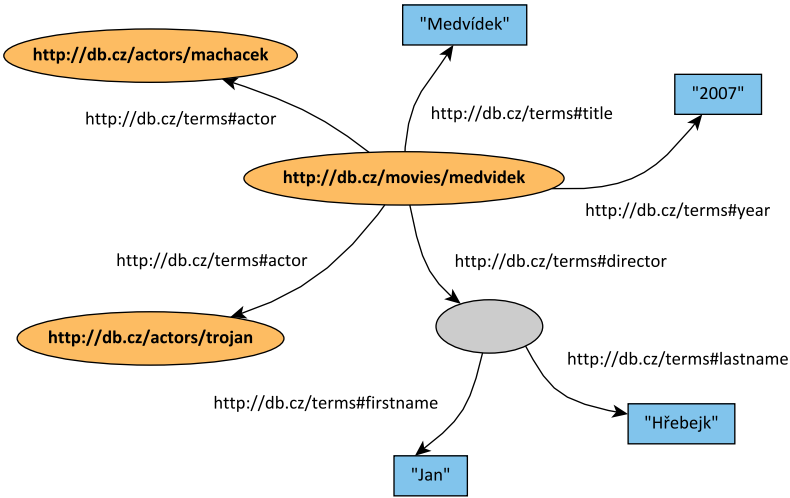
- **Blank nodes** (anonymous resources)
 - Allow to express statements about resources without explicitly naming (identifying) them
- **Blank node identifiers** only have local scope of validity
 - E.g. within a given file, query expression, ...
- Particular syntax depends on a serialization format
 - E.g.: `_:node18`

Data Model

Directed labeled multigraph

- Vertices
 - **One vertex** for each distinct **subject and object**
- Edges
 - **One edge** for each individual **triple**
 - Edges are directed $subject \xrightarrow{predicate} object$
 - Property names (predicate IRIs) are used as edge labels

Example



Serialization

Available approaches

- **N-Triples** notation
 - <https://www.w3.org/TR/n-triples/>
- **Turtle** notation (*Terse RDF Triple Language*)
 - <https://www.w3.org/TR/turtle/>
- **RDF/XML** notation
 - XML syntax for RDF
 - <https://www.w3.org/TR/rdf-syntax-grammar/>
- **JSON-LD** notation
 - JSON-based serialization for Linked Data
 - <https://www.w3.org/TR/json-ld/>
- ...

N-Triples Notation

RDF **N-Triples** notation = *A line-based syntax for an RDF graph*

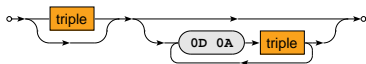
- Simple, line-based, plain text format
- File extension: ***.rdf**
- <https://www.w3.org/TR/n-triples/>

Example

- *Already presented...*

Document

- Statements are terminated by dots, delimited by EOL



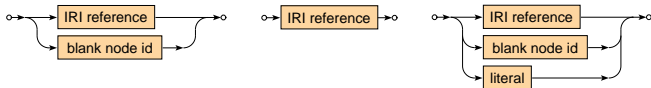
N-Triples Notation

Statement

- Individual triple components are delimited by spaces



Triple components: subject, predicate, object



N-Triples Notation

IRI reference

- IRIs are enclosed in angle brackets

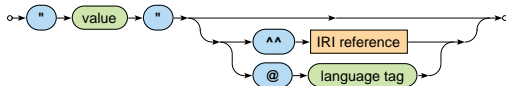


Blank node identifier



Literal

- Literals are enclosed in double quotes



Turtle Notation

Turtle = *Terse RDF Triple Language*

- Compact format with various abbreviations
- File extension: ***.ttl**
- Content type: text/turtle
- <https://www.w3.org/TR/turtle/>

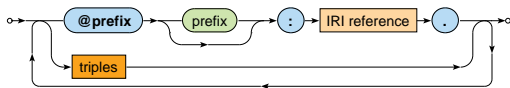
Example

```
@prefix i: <http://db.cz/terms#> .
@prefix m: <http://db.cz/movies/> .
@prefix a: <http://db.cz/actors/> .
m:medvidek
  i:title "Medvídek" ;
  i:actor a:machacek , a:trojan ;
  i:year 2007 ;
  i:director [ i:firstname "Jan" ; i:lastname "Hřebejk" ] .
```

Turtle Notation

Document

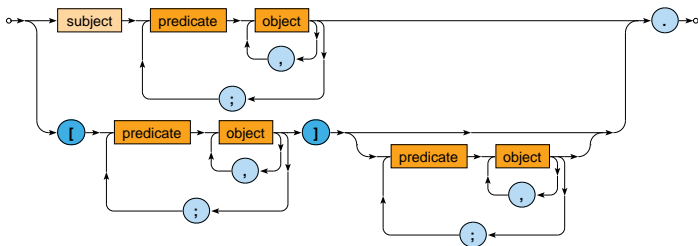
- Contains a **sequence of triples and/or declarations**
- Prefix declarations
 - **Prefixed names** can then be used instead of full IRI references
- Groups of triples
 - Individual groups are terminated by dots



Turtle Notation

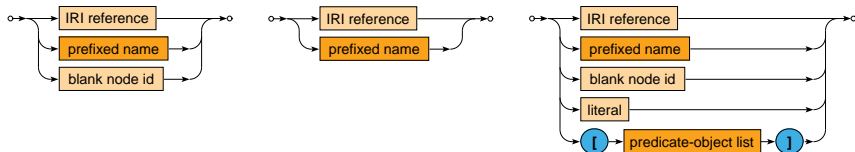
Triples

- Triples sharing the same subject and object or at least the same subject can be *grouped* together
 - **object** list for a **shared subject and predicate**
 - **predicate-object** list for a **shared subject**
- Brackets can be used to define blank nodes

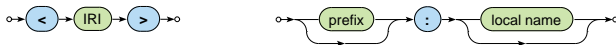


Turtle Notation

Triple components: subject, predicate, object



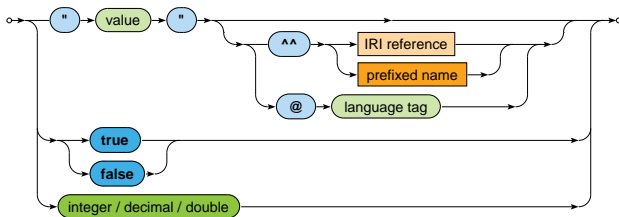
IRI reference / prefixed name



Turtle Notation

Literal

- Traditional literals
+ new abbreviated forms of numeric and boolean literals



Example

Example revisited

```
@prefix i: <http://db.cz/terms#> .
@prefix m: <http://db.cz/movies/> .
@prefix a: <http://db.cz/actors/> .
m:medvidek
  i:title "Medvídek" ;
  i:actor a:machacek , a:trojan ;
  i:year 2007 ;
  i:director [ i:firstname "Jan" ; i:lastname "Hřebejk" ] .
```

RDF Conclusion

RDF statements

- Subject, predicate, and object components

Schema languages

- RDFS (*RDF Schema*)
- OWL (*Web Ontology Language*)

Query languages

- SPARQL (*SPARQL Protocol and RDF Query Language*)

SPARQL Query Language

SPARQL

SPARQL Query Language

- Query language for RDF data
 - **Graph patterns**, optional graph patterns, **subqueries**, negation, **aggregation**, value constructors, ...
- Versions: 1.0 (2008), **1.1** (2013)
- W3C recommendations
 - <https://www.w3.org/TR/sparql11-query/>
 - Altogether 11 parts: query language, update facility, federated queries, protocol, result formats, ...

Sample Data

Graph of **movies** <http://db.cz/movies>

```
@prefix i: <http://db.cz/terms#> .
@prefix m: <http://db.cz/movies/> .
@prefix a: <http://db.cz/actors/> .

m:vratnelahve
  rdf:type i:Movie ; i:title "Vratné lahve" ;
  i:year 2006 ;
  i:actor a:sverak , a:machacek .

m:samotari
  rdf:type i:Movie ; i:title "Samotáři" ;
  i:year 2000 ;
  i:actor a:schneiderova , a:trojan , a:machacek .

m:medvidek
  rdf:type i:Movie ; i:title "Medvídek" ;
  i:year 2007 ;
  i:actor a:machacek , a:trojan ;
  i:director "Jan Hřebejk" .

m:zelary
  rdf:type i:Movie .
```

Sample Data

Graph of **actors** <http://db.cz/actors>

```
@prefix i: <http://db.cz/terms#> .
@prefix a: <http://db.cz/actors/> .
a:trojan
  rdf:type i:Actor ;
  i:firstname "Ivan" ; i:lastname "Trojan" ;
  i:year 1964 .
a:machacek
  rdf:type i:Actor ;
  i:firstname "Jiří" ; i:lastname "Macháček" ;
  i:year 1966 .
a:schneiderova
  rdf:type i:Actor ;
  i:firstname "Jitka" ; i:lastname "Schneiderová" ;
  i:year 1973 .
a:sverak
  rdf:type i:Actor ;
  i:firstname "Zdeněk" ; i:lastname "Svěrák" ;
  i:year 1936 .
```

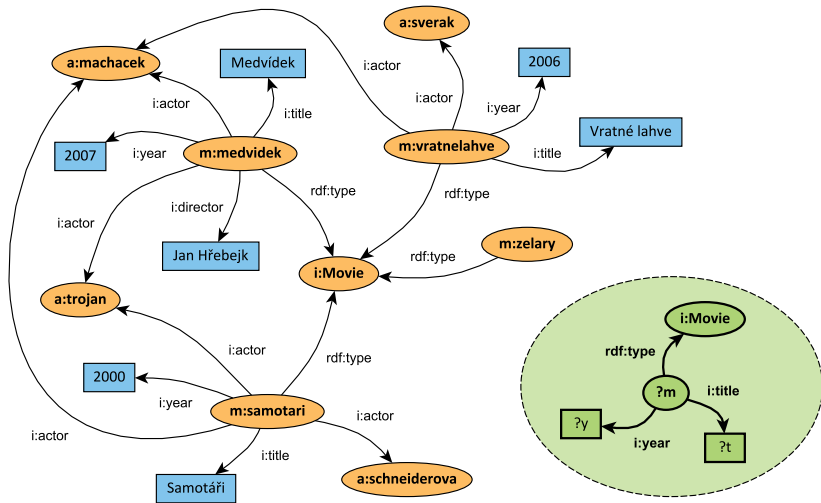

Sample Query

Find all movies, return their titles and years they were filmed

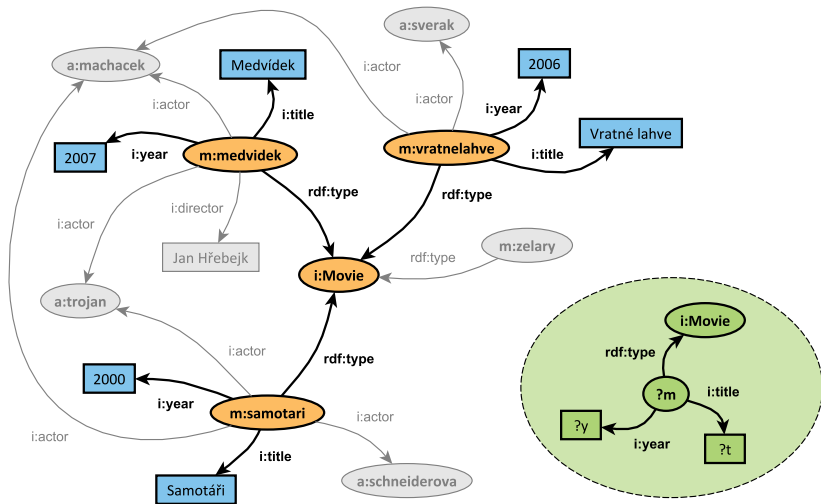
```
PREFIX i: <http://db.cz/terms#>
SELECT ?t ?y
FROM <http://db.cz/movies>
WHERE
{
  ?m rdf:type i:Movie ;
    i:title ?t ;
    i:year ?y .
}
ORDER BY ?y
```

?t	?y
Samotáři	2000
Vratné lahve	2006
Medvídek	2007

Sample Query



Sample Query



Graph Pattern Matching

Graph patterns

- **Basic graph pattern**
 - Based on ordinary **triples with variables**
 - ?variable or \$variable
- More complicated graph patterns
 - E.g. **group**, **optional**, **minus**, ...

Graph pattern matching

- Our goal is to find all **subgraphs of the data graph that are matched by the query graph pattern**
 - I.e. subgraphs of the data graph that are identical to the query graph pattern with variables substituted by particular terms
- One **matching subgraph** = one **solution** = one **row of a table**

Graph Pattern Matching

Query result = **solution sequence** = ordered multiset of solutions

?t	?y
Samotáři	2000
Vratné lahve	2006
Medvídek	2007

```
{  
  { (?t, "Samotáři"), (?y, "2000") },  
  { (?t, "Vratné lahve"), (?y, "2006") },  
  { (?t, "Medvídek"), (?y, "2007") }  
}
```

Solution = set of variable bindings

?t	?y
Samotáři	2000

```
{ (?t, "Samotáři"), (?y, "2000") }
```

Variable binding = pair of a variable name and a value it is assigned

?t
Samotáři

```
(?t, "Samotáři")
```

Graph Pattern Matching

Compatibility of solutions

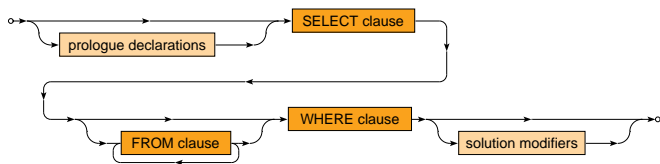
- Two solutions are mutually compatible if and only if all the variables they share are pairwise bound to identical values

Examples

- Compatible solutions
 - `{ (?m, m:samotari), (?t, "Samotáři") }`
`{ (?m, m:samotari), (?y, "2000") }`
- Incompatible solutions
 - `{ (?m, m:samotari), (?t, "Samotáři") }`
`{ (?m, m:medvidek), (?y, "2007") }`

Select Queries

SELECT queries



- **Prologue declarations** – PREFIX, BASE
- **Main clauses**
 - **SELECT** – **variables to be projected**
 - **FROM** – **data graphs to be queried**
 - **WHERE** – **graph patterns to be matched**
- **Solution modifiers** – ORDER BY, ...

Prologue Declarations

Prologue declarations

- Allow to simplify IRI references by declaring **base IRIs**



BASE clause

- At most **one base IRI** can be defined
- All **relative IRI references** are then related to this base IRI

PREFIX clause

- Several base IRIs** are defined, each is associated with a name
 - These names must be distinct, one of them can be empty
- All **prefixed names** are then related to the respective base IRIs

Prologue Declarations

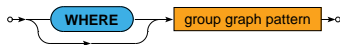
Examples

- When `BASE <http://db.cz/>` is defined, then a relative IRI reference `terms#Movie` is interpreted as `http://db.cz/terms#Movie`
- When `PREFIX i: <http://db.cz/>` is defined, then a prefixed name `i:terms#Movie` is interpreted as `http://db.cz/terms#Movie`

Where Clause

WHERE clause

- Describes one **group graph pattern**



Types of graph patterns

- **Basic** – triple patterns to be matched
- **Group** – set of graph patterns to be matched
- **Optional** – graph pattern to be matched only if possible
- **Alternative** – two or more alternative graph patterns
- ...

Graph patterns can be inductively combined into complex ones

Basic Graph Pattern

Basic graph pattern (block of triples)

One or more triple patterns to be all matched

- Ordinary **triples** separated by `.`
 - E.g. `s p1 o1 . s p1 o2 . s p2 o3 .`
- May contain variables
 - E.g. `?var` or `$var`
- Turtle abbreviations also permitted
 - **Object lists** using `,` or **predicate-object lists** using `;`
 - E.g. `s p1 o1 , o2 ; p2 o3 .`
 - **Blank nodes** using `[]`
 - Act as non-selectable variables
 - I.e. do not enforce to be matched only by blank nodes in data!

Basic Graph Pattern

Interpretation

- **All the involved triples must be matched**
 - I.e. we combine them as if they were in conjunction
 - More precisely...
 - Each triple pattern is evaluated to its solution sequence
 - **All combinations of compatible solutions** are then found
- Note that all the variables need to be bound
 - I.e. if any of the involved variables cannot be bound at all, then the entire basic graph pattern cannot be matched!

Basic Graph Pattern: Example

Titles and years of all movies

```
PREFIX i: <http://db.cz/terms#>
SELECT ?t ?y
FROM <http://db.cz/movies>
WHERE
{
  ?m rdf:type i:Movie . # triple 1
  ?m i:title ?t .       # triple 2
  ?m i:year ?y .       # triple 3
}
```

?t	?y
Vratné lahve	2006
Samotáři	2000
Medvídek	2007

Basic Graph Pattern: Example

$\llbracket t_1 \rrbracket =$

?m
m:vratnelahve
m:samotari
m:medvidek
m:zelary

$\llbracket t_2 \rrbracket =$

?m	?t
m:vratnelahve	Vratné lahve
m:samotari	Samotáři
m:medvidek	Medvídek

$\llbracket t_3 \rrbracket =$

?m	?y
m:vratnelahve	2006
m:samotari	2000
m:medvidek	2007

?t	?y
Vratné lahve	2006
Samotáři	2000
Medvídek	2007



Equality of Terms

IRIs

Literals

- Plain values must be identical
- And when **types / language tags** are specified, these must be identical as well
 - E.g.: "Medvídek"@cs != "Medvídek"

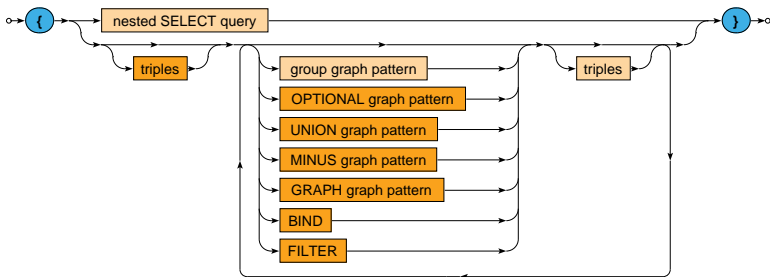
Blank nodes

- Blank nodes in query patterns act as **non-selectable variables**
- Labels of blank nodes in data graphs / query graph patterns / query results **may not refer to the same nodes despite being the same**
 - I.e. the scope of validity is always local only

Group Graph Pattern

Group graph pattern

Set of graph patterns to be all matched



Group Graph Pattern

Two modes

- **Nested SELECT query**
 - Only with SELECT and WHERE clauses and solution modifiers
i.e. without FROM clause
- **Set of graph patterns interleaved by triple blocks**

Interpretation

- **All the involved graph patterns must be matched**
 - I.e. we combine them as if they were in conjunction

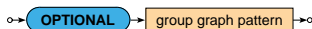
Notes

- Empty group patterns {} are also allowed

Optional Graph Pattern

OPTIONAL graph pattern

One group graph pattern is tried to be matched



Interpretation

- **When the optional part does not match, it creates no bindings but does not eliminate the solution**

Optional Graph Pattern: Example

Movies together with their directors when possible

```
PREFIX i: <http://db.cz/terms#>
SELECT ?t ?y ?d
FROM <http://db.cz/movies>
WHERE
{
  ?m rdf:type i:Movie ;
    i:title ?t ;
    i:year ?y .
  OPTIONAL { ?m i:director ?d . }
}
```

?t	?y	?d
Vratné lahve	2006	
Samotáři	2000	
Medvídek	2007	Jan Hřebejk

Alternative Graph Pattern

UNION graph pattern

Two or more group graph patterns are to be matched



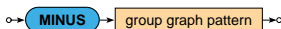
Interpretation

- Standard set **union** of the involved query results

Minus Graph Pattern

MINUS graph pattern

One group graph pattern removing compatible solutions



Interpretation

- **Solutions of the first pattern are preserved if and only if they are not compatible with any solution of the second pattern**
 - I.e. minus graph pattern does not correspond to the standard set minus operation!

Minus Graph Pattern: Example

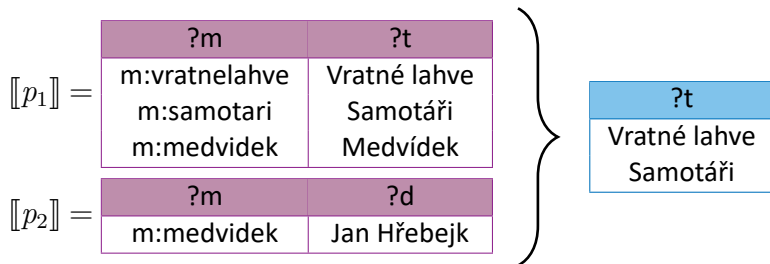
Titles of movies that have no director

```
PREFIX i: <http://db.cz/terms#>
SELECT ?t
FROM <http://db.cz/movies>
WHERE
{
  ?m rdf:type i:Movie ;
    i:title ?t . # pattern 1
  MINUS { ?m rdf:type i:Movie ; i:director ?d . } # pattern 2
}
```

?t

Vratné lahve
Samotáři

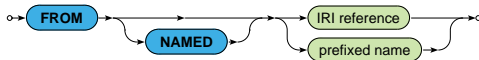
Minus Graph Pattern: Example



From Clause

FROM clause

- Defines data graphs to be queried



Dataset = collection of graphs to be queried

- One **default graph**
 - Merge of all the declared graphs from unnamed FROM clauses
 - Empty when no unnamed FROM clause is provided
- Zero or more **named graphs**

Active graph = used for the evaluation of graph patterns

- The default graph unless changed using GRAPH graph pattern

From Clause: Example

Names of actors who played in *Medvídek* movie

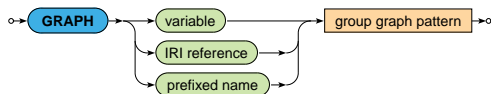
```
PREFIX i: <http://db.cz/terms#>
PREFIX m: <http://db.cz/movies/>
SELECT ?f ?l
FROM <http://db.cz/movies>
FROM <http://db.cz/actors>
WHERE
{
  m:medvidek i:actor ?a .
  ?a i:firstname ?f ; i:lastname ?l .
}
```

?f	?l
Jiří	Macháček
Ivan	Trojan

Graph Graph Pattern

GRAPH graph pattern

Pattern evaluated with respect to a particular named graph



- **Changes the active graph** for a given group graph pattern
 - `GRAPH <http://db.cz/actors> { ... }`
- We can also consider all the named graphs
 - `GRAPH ?g { ... }`

Graph Graph Pattern: Example

Names of actors who played in *Medvídek* movie

```
PREFIX i: <http://db.cz/terms#>
PREFIX m: <http://db.cz/movies/>
SELECT ?f ?l
FROM <http://db.cz/movies>
FROM NAMED <http://db.cz/actors>
WHERE
{
  m:medvidek i:actor ?a .
  GRAPH <http://db.cz/actors> {
    ?a i:firstname ?f ; i:lastname ?l .
  }
}
```

?f	?l
Jiří	Macháček
Ivan	Trojan

Variable Assignments

BIND graph pattern

Explicitly assigns a value to a given variable

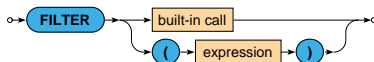


- This variable must not yet be bound!

Filter Constraints

FILTER constraints

Impose constraints on variables and their values



- **Only solutions satisfying the given condition are preserved**
- Does not create any new variable bindings!
- Always applied on the entire group graph pattern
i.e. evaluated at the very end

Filter Constraints: Example

Movies filmed in 2005 or later where *Ivan Trojan* played

```
PREFIX i: <http://db.cz/terms#>
PREFIX a: <http://db.cz/actors/>
SELECT ?t ?y
FROM <http://db.cz/movies>
WHERE
{
  ?m rdf:type i:Movie ;
    i:title ?t ;
    i:year ?y .
  FILTER (
    (?y >= 2005) &&
    EXISTS { ?m i:actor a:trojan . }
  )
}
```

?t	?y
Medvídek	2007

Filter Constraints

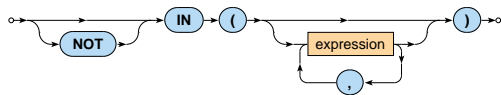
Relational expressions

- **Comparisons**

- =, !=, <, <=, =>, >
- Unbound variable < blank node < IRI < literal

- **Set membership tests**

- IN and NOT IN



Numeric expressions

- Unary / binary **arithmetic operators** +, -, *, /

Filter Constraints

Primary expressions

- **Literals** – numeric, boolean, RDF triples
- **Variables**
- Built-in calls
- Parentheses

Boolean expressions

- Logical connectives
 - Conjunction `&&`, disjunction `||`, negation `!`
- **3-value logic** because of unbound variables (NULL values)
 - true, false, error

Filter Constraints

Built-in calls

- **Term accessors**
 - STR – lexical form of an IRI or literal
 - LANG – language tag of a literal
 - DATATYPE – data type of a literal
- **Variable tests**
 - BOUND – true when a variable is bound to a value
 - isIRI, isBLANK, isLITERAL

Filter Constraints

Built-in calls

- **Existence tests**

- EXISTS

- True when a provided group graph pattern is evaluated to at least one solution

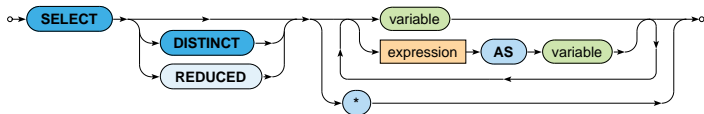
- NOT EXISTS



Select Clause

SELECT clause

- Enumerates variables to be included in the query result



- Asterisk * selects all the variables

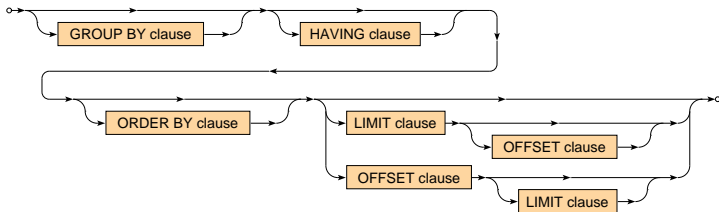
Solution modifiers

- DISTINCT – **duplicate solutions are removed**
- REDUCED – some duplicate solutions may be removed (implementation-dependent behavior)

Solution Modifiers

Solution modifiers – modify the entire solution sequence

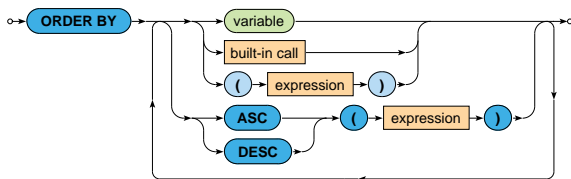
- Aggregation
 - GROUP BY and HAVING
- Ordering
 - ORDER BY
 - LIMIT and OFFSET



Solution Modifiers

ORDER BY clause

- Defines the order of solutions within the query result

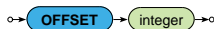


- `ASC(...)` = **ascending** (default)
- `DESC(...)` = **descending**

Solution Modifiers

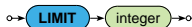
OFFSET clause

- **Skips a certain number of solutions** in the query result



LIMIT clause

- **Limits the number of solutions** in the query result



Solution Modifiers: Example

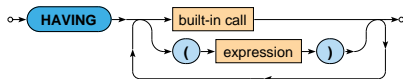
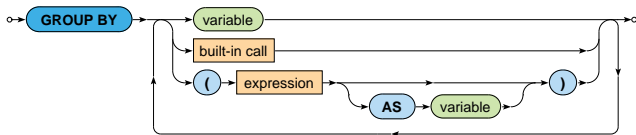
```
PREFIX i: <http://db.cz/terms#>
SELECT ?t ?y
FROM <http://db.cz/movies>
WHERE
  {
    ?m rdf:type i:Movie ;
      i:title ?t ;
      i:year ?y .
  }
ORDER BY DESC(?y) ASC(?t)
OFFSET 1
LIMIT 5
```

?t	?y
Vratné lahve	2006
Samotáři	2000

Aggregation

GROUP BY + HAVING clauses

- Standard aggregation over a solution sequence



Aggregation: Example

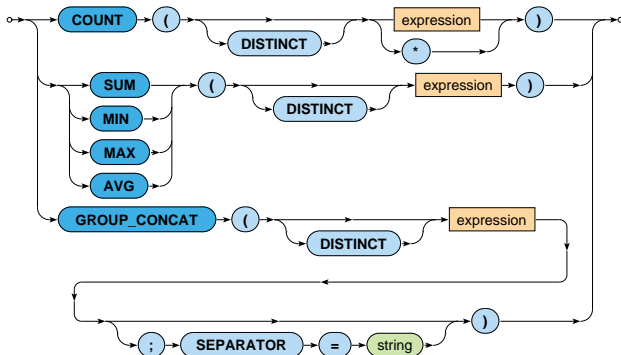
Numbers of actors in movies with at most 2 actors

```
PREFIX i: <http://db.cz/terms#>
SELECT ?t (COUNT(?a) AS ?c)
FROM <http://db.cz/movies>
WHERE
{
  ?m rdf:type i:Movie ;
  i:title ?t ;
  i:actor ?a .
}
GROUP BY ?m ?t
HAVING (?c <= 2)
ORDER BY ?c ?t
```

?t	?c
Medvídek	2
Vratné lahve	2

Aggregation

Aggregate functions



Query Forms

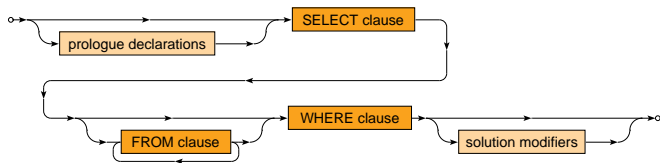
Query forms

- **SELECT**
 - Finds solutions matching a provided graph pattern
- **ASK**
 - Checks whether at least one solution exists
- **DESCRIBE**
 - Retrieves a graph with data about selected resources
- **CONSTRUCT**
 - Creates a new graph according to a provided pattern

Select Query Form

SELECT query form

Finds solutions matching a provided graph pattern



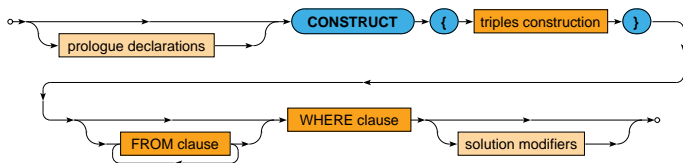
Result

- **Solution sequence** = ordered multiset of solutions

Construct Query Form

CONSTRUCT query form

Creates a new graph according to a provided pattern



Result

- **RDF graph** constructed according to a group graph pattern
 - Unbound or invalid triples are not involved

Construct Query Form: Example

```
PREFIX i: <http://db.cz/terms#>
CONSTRUCT
  {
    ?a i:name concat(?f, " ", ?l) .
  }
FROM <http://db.cz/actors>
WHERE
  {
    ?a rdf:type i:Actor ;
      i:firstname ?f ;
      i:lastname ?l .
  }
```

```
<http://db.cz/actors/trojan> i:name "Ivan Trojan" .
<http://db.cz/actors/machacek> i:name "Jiří Macháček" .
<http://db.cz/actors/schneiderova> i:name "Jitka Schneiderová" .
<http://db.cz/actors/sverak> i:name "Zdeněk Svěrák" .
```


Lecture Conclusion

SPARQL

- **Query forms**
 - SELECT, ASK, DESCRIBE, CONSTRUCT
- **Graph patterns**
 - Basic, group, optional, alternative, minus
 - Variable assignments
 - Filters
- **Solution modifiers**
 - DISTINCT, REDUCED
 - GROUP BY, HAVING
 - ORDER BY, LIMIT, OFFSET