TECHNISCHE UNIVERSITÄT DRESDEN

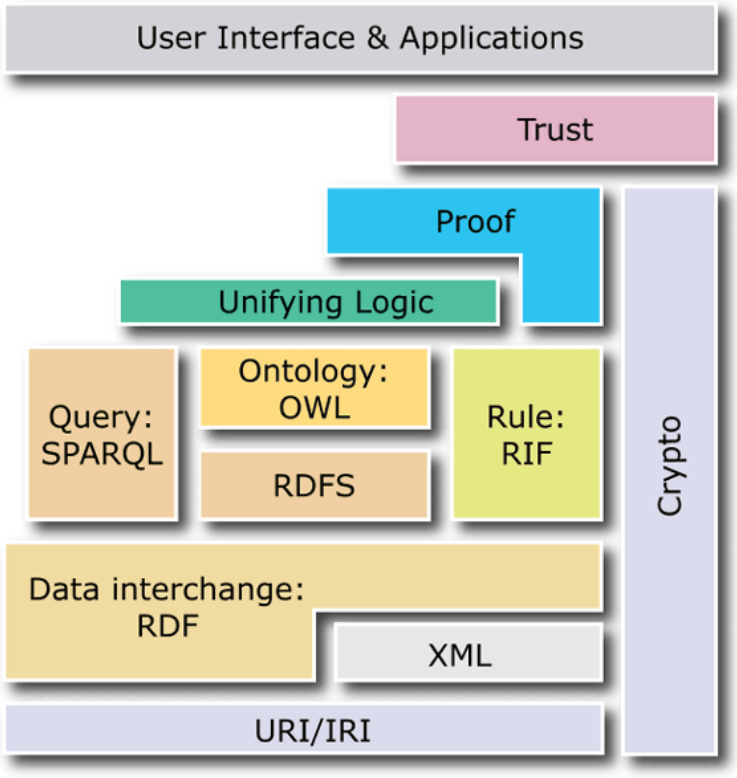
FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

SPARQL Syntax & Intuition

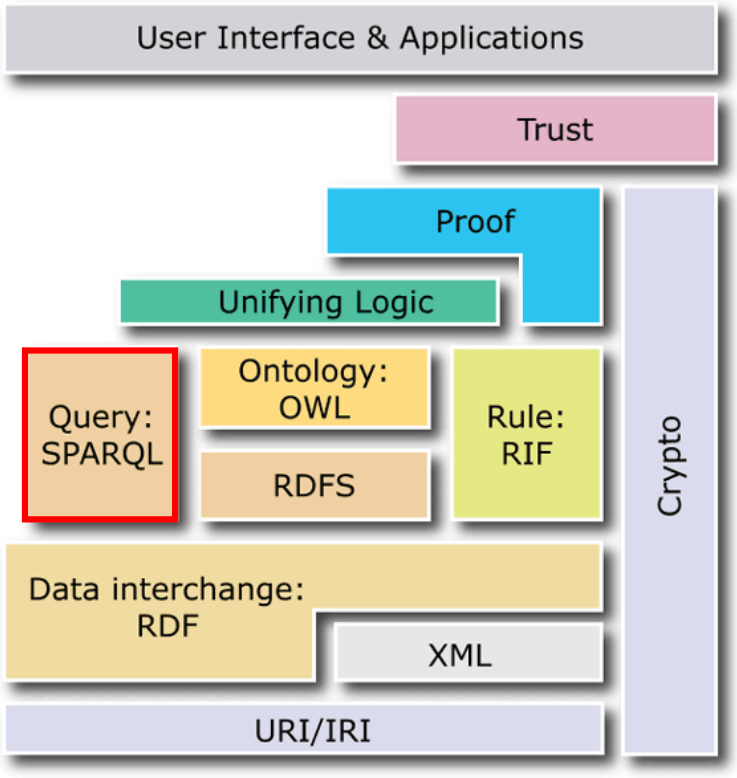
Sebastian Rudolph

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

The SPARQL Query Language



The SPARQL Query Language



Agenda

1. Introduction and Motivation
2. Simple SPARQL Queries
3. Complex Graph Patterns
4. Filters
5. Solution Modifiers
6. Conclusions & Outlook

Agenda

1 Introduction and Motivation

2 Simple SPARQL Queries

3 Complex Graph Patterns

4 Filters

5 Solution Modifiers

6 Conclusions & Outlook

Query Languages for the Semantic Web?

How can we access information specified in RDF(S) or OWL?

RDF(S) Data

- Simple Entailment
- RDF-Entailment
- RDFS-Entailment

“Is one RDF graph a consequence of another one?”

Query Languages for the Semantic Web?

How can we access information specified in RDF(S) or OWL?

RDF(S) Data

- Simple Entailment
- RDF-Entailment
- RDFS-Entailment

"Is one RDF graph a consequence of another one?"

OWL ontologies

- Logical Entailment

"Does an OWL ontology entail a subsumption relation between two classes?"
"What are the instances of a class in an OWL ontology?"

Do OWL and RDF(S) not suffice?

Even OWL is too weak to formulate queries

- “Which strings does the ontology specify in German?”
- “Which properties relate two given individuals?”
- “Which pairs of persons have a common parent?”

$\rightsquigarrow$ Expressible neither in RDF nor in OWL

Do OWL and RDF(S) not suffice?

Even OWL is too weak to formulate queries

- "Which strings does the ontology specify in German?"
- "Which properties relate two given individuals?"
- "Which pairs of persons have a common parent?"

$\rightsquigarrow$ Expressible neither in RDF nor in OWL

Requirements:

- High expressivity for describing the queried information
- Possibility of formatting, restricting, and manipulating the results

Requirements for a Query Language

- High expressivity for describing the required data
- Support for selecting, manipulating, and formatting of the results
- More?

Agenda

1 Introduction and Motivation

2 Simple SPARQL Queries

3 Complex Graph Patterns

4 Filters

5 Solution Modifiers

6 Conclusions & Outlook

SPARQL

SPARQL (pronounced sparkle) stands for
SPARQL Protocol And RDF Query Language

- W3C Specification since 2008
- Extension to SPARQL 1.1 since 2013
- Query language to query RDF graphs
- Very practice relevant

Parts of the SPARQL 1.0 specification

- Query: The syntax and semantics of the query language
- Query Results XML Format: how to display results in XML
- Protocol for RDF: conveying SPARQL queries to a SPARQL query processing service and returning the results

Parts of the SPARQL 1.1 Specification

- Query: extends the language constructs for SPARQL queries
- Update: modify an RDF graph (addition, deletion)
- Graph Store HTTP Protocol: HTTP operations for managing a collection of graphs
- Entailment Regimes: query results with inferences
- Service Description: method for discovering, and vocabulary for describing SPARQL services
- Federation Extensions: executing distributed queries
- Query Results JSON Format: query results in JSON format
- Query Results CSV, TSV Format: comma and tab separated results format

Simple Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

Simple Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

- The condition of the WHERE clause is called a query pattern

Simple Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

- The condition of the WHERE clause is called a query pattern
- The triples (possibly) with variables are called a basic graph pattern (BGP)
  ⇝ BGPs use the Turtle syntax for RDF
  ⇝ BGPs can contain variables (`?variable`/`$variable`)

Simple Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

- The condition of the WHERE clause is called a query pattern
- The triples (possibly) with variables are called a basic graph pattern (BGP)
  ⇝ BGPs use the Turtle syntax for RDF
  ⇝ BGPs can contain variables (`?variable`/`$variable`)
- Abbreviated IRIs are possible (`PREFIX`)

Simple Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

- The condition of the WHERE clause is called a query pattern
- The triples (possibly) with variables are called a basic graph pattern (BGP)
  ⇝ BGPs use the Turtle syntax for RDF
  ⇝ BGPs can contain variables (`?variable`/`$variable`)
- Abbreviated IRIs are possible (`PREFIX`)
- Query result for the selected variables (`SELECT`)

Simple Query – Result

BGP: `{?x foaf:name ?name . ?x foaf:mbox ?mbox}`

```
@prefix foaf: http://xmlns.com/foaf/0.1/ .
_:a foaf:name "Birte Glimm" ;
    foaf:mbox "b.glimm@googlemail.com" ;
    foaf:icqChatID "b.glimm" ;
    foaf:aimChatID "b.glimm" .
_:b foaf:name "Sebastian Rudolph" ;
    foaf:mbox <mailto:rudolph@kit.edu> .
_:c foaf:name "Pascal Hitzler" ;
    foaf:aimChatID "phi" .
foaf:icqChatID rdfs:subPropertyOf foaf:nick .
foaf:name rdfs:domain foaf:Person .
```

Simple Query – Result

BGP: `{?x foaf:name ?name . ?x foaf:mbox ?mbox}`

```
@prefix foaf: http://xmlns.com/foaf/0.1/ .
_:a foaf:name "Birte Glimm" ;
    foaf:mbox "b.glimm@googlemail.com" ;
    foaf:icqChatID "b.glimm" ;
    foaf:aimChatID "b.glimm" .
_:b foaf:name "Sebastian Rudolph" ;
    foaf:mbox <mailto:rudolph@kit.edu> .
_:c foaf:name "Pascal Hitzler" ;
    foaf:aimChatID "phi" .
foaf:icqChatID rdfs:subPropertyOf foaf:nick .
foaf:name rdfs:domain foaf:Person .
```

BGP matching results:

x	name	mbox
_:a	"Birte Glimm"	"b.glimm@googlemail.com"
_:b	"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Simple Query – Result

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

BGP matching results:

x	name	mbox
_:a	"Birte Glimm"	"b.glimm@googlemail.com"
_:b	"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Query results:

name	mbox
"Birte Glimm"	"b.glimm@googlemail.com"
"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Basic Graph Patterns

The most basic query patterns are basic graph patterns

- Set of RDF triples in Turtle syntax
- Turtle abbreviations (such as `,` and `;`) allowed
- Variables are prefixed by `?` or `$` (`?x` identifies the same variable as `$x`)
- Variables can appear in subject, predicate, and object position

Basic Graph Patterns

The most basic query patterns are basic graph patterns

- Set of RDF triples in Turtle syntax
- Turtle abbreviations (such as `,` and `;`) allowed
- Variables are prefixed by `?` or `$` (`?x` identifies the same variable as `$x`)
- Variables can appear in subject, predicate, and object position

permitted $\neq$ readable:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?rf456df ?ac66sB
WHERE { ?h4dF8Q foaf:name ?rf456df .
        ?h4dF8Q foaf:mbox ?ac66sB }
```

(semantically equivalent to the previous query)

Blank Nodes

What meaning do blank nodes have in SPARQL?

Blank nodes in query patterns:

- Permitted as subject or object (as in RDF)
- Arbitrary ID, but reuse in different BGPs within one query not permitted
- Act like variables, but cannot be selected

Blank Nodes

What meaning do blank nodes have in SPARQL?

Blank nodes in query patterns:

- Permitted as subject or object (as in RDF)
- Arbitrary ID, but reuse in different BGPs within one query not permitted
- Act like variables, but cannot be selected

Blank nodes in results:

- Placeholder for unknown elements
- Arbitrary IDs (possibly different from the IDs in the input RDF graph), but repeated occurrences in results denote the same element:

subj	value
_:a	"for"
_:b	"example"

subj	value
_:y	"for"
_:g	"example"

subj	value
_:z	"for"
_:z	"example"

Datasets and `FROM (NAMED)`

- No `FROM` clause is required
- Each SPARQL service specifies a dataset of one default graph and zero or more named graphs

No `FROM` clause

$\rightsquigarrow$ evaluation over the default graph

`FROM NAMED` in combination with the `GRAPH` keyword

$\rightsquigarrow$ evaluation over a named graph

`FROM` clause

$\rightsquigarrow$ creation of a fresh default graph for the query

Example for Named Graphs

Query with `FROM NAMED` clause

```
SELECT ?g ?name ?mbox
FROM NAMED <http://ex.org/a>
FROM NAMED <http://ex.org/b>
WHERE {
  GRAPH ?g
    { ?x foaf:name ?name.
      ?x foaf:mbox ?mbox }
}
```

Datatypes

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:ex1  ex:p  "test" .
ex:ex2  ex:p  "test"^^xsd:string .
ex:ex3  ex:p  "test"@en .
ex:ex4  ex:p  "42"^^xsd:integer .
```

Which matches does the following BGP have?

```
{   ?subject  <http://example.org/p>  "test" . }
```

Datatypes

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:ex1  ex:p  "test" .
ex:ex2  ex:p  "test"^^xsd:string .
ex:ex3  ex:p  "test"@en .
ex:ex4  ex:p  "42"^^xsd:integer .
```

Which matches does the following BGP have?

```
{   ?subject  <http://example.org/p>  "test" . }
```

⇝ `ex:ex1` is the only result
⇝ Exact match for the datatypes is required

But: Abbreviations for numerical values allowed

```
{   ?subject  <http://example.org/p>  42 . }
```

⇝ The datatype is determined from the syntactic form
`xsd:integer` (42), `xsd:decimal` (42.2), `xsd:double` (1.0e6)

Agenda

1 Introduction and Motivation

2 Simple SPARQL Queries

3 Complex Graph Patterns

4 Filters

5 Solution Modifiers

6 Conclusions & Outlook

Group Graph Patterns

Basic graph patterns can be grouped by {...}.

Example:

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?author
WHERE
{ { ?book ex:publishedBy <http://springer.com>.
    ?book ex:titel ?titel . }
  { }
  ?book ex:author ?author .
}
```

$\rightsquigarrow$ Only useful in combination with additional constructors

Optional Patterns

The keyword `OPTIONAL` permits the specification of optional parts for a graph pattern.

Example:

```
{ ?book    ex:publishedBy  <http://springer.com> .
  OPTIONAL { ?book ex:titel ?titel . }
  OPTIONAL { ?book ex:author ?author . }
}
```

Optional Patterns

The keyword `OPTIONAL` permits the specification of optional parts for a graph pattern.

Example:

```
{ ?book    ex:publishedBy  <http://springer.com> .
  OPTIONAL { ?book ex:titel ?titel . }
  OPTIONAL { ?book ex:author ?author . }
}
```

$\rightsquigarrow$ Parts of the query result can be unbound:

book	titel	author
<http://ex.org/book1>	"Titel1"	<http://ex.org/author1>
<http://ex.org/book2>	"Titel2"	
<http://ex.org/book3>	"Titel3"	_:a
<http://ex.org/book4>		_:a
<http://ex.org/book5>		

Alternative Patterns

The keyword `UNION` allows for specifying alternative parts for a pattern.

Example:

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

$\rightsquigarrow$ Results corresponds to the union of the results for the first BGP with the results for one of the additional BGPs

Remark: Identical variables within different `UNION` patterns do not influence each other

Excercise

Data

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://ex.org/> .
_:a  dc10:title     "SPARQL Query Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:b  ex:level "beginners" .
```

Write a query that selects the title (`dc10:title` or `dc11:title`) and, where given, the level (`ex:level`)

Solution

Solution

Query

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>
PREFIX ex:  <http://ex.org/>
SELECT ?title ?level
WHERE {
    { { ?book dc10:title ?title }
      UNION { ?book dc11:title ?title }
    } OPTIONAL { ?book ex:level ?level }
}
```

title	level
"SPARQL Query Tutorial"	
"SPARQL Protocol Tutorial"	"beginners"

Combination of Optional and Alternatives (1)

How can we understand the combination of `OPTIONAL` and `UNION`?

Example

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

- The union of two patterns with appended optional pattern or
- The union of two patterns where the second one has an optional part?

Combination of Optional and Alternatives (1)

How can we understand the combination of `OPTIONAL` and `UNION`?

Example

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

- The union of two patterns with appended optional pattern or ✓
- The union of two patterns where the second one has an optional part?

Combination of Optional and Alternatives (1)

Example

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

is equivalent to

Example with explicit grouping

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

Combination of Optional and Alternatives (2)

General Rules:

- `OPTIONAL` always applies to one pattern group, which is specified to right of the keyword `OPTIONAL`.
- `OPTIONAL` and `UNION` have equal precedence and apply to all parts to the left of the keyword (left associative).

Combination of Optional and Alternatives (3)

Example

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}
              OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
```

Combination of Optional and Alternatives (3)

Example

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}
             OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
```

Can be understood as:

Equivalent example with explicit grouping

```
  { { { { {s1 p1 o1} OPTIONAL {s2 p2 o2}
        } UNION {s3 p3 o3}
      } OPTIONAL {s4 p4 o4}
    } OPTIONAL {s5 p5 o5}
  }
```

Agenda

1 Introduction and Motivation

2 Simple SPARQL Queries

3 Complex Graph Patterns

4 Filters

5 Solution Modifiers

6 Conclusions & Outlook

Why Filters?

Many queries are not expressible, even with complex query patterns:

- “Which persons are between 18 and 23 years old?”
- “The surname of which person contains a hyphen?”
- “Which texts in the ontology are specified in German?”

$\rightsquigarrow$ Filter as a general mechanism for such expressions

Filter in SPARQL

Example:

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

- Keyword `FILTER`, followed by a filter expression in brackets
- Filter conditions evaluate to truth values (and possibly errors)
- Many filter functions are not specified by RDF
  $\rightsquigarrow$ Functions partly taken from the XQuery/XPath-standard for XML

Filter Functions: Comparisons

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

- Comparison of literals according to the natural order
- Support for numerical datatypes, `xsd:dateTime`, `xsd:string` (alphabetical order), `xsd:Boolean` (1 > 0)
- For other types or RDF elements only `=` und `!=` available
- No comparison between literals with incompatible types (e.g., `xsd:string` and `xsd:integer`)

Filter Functions: Arithmetic

Arithmetic operators: `+`, `-`, `*`, `/`

- Support for numerical datatypes
- Used to combine values in filter conditions

Example

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

Filter Functions: Special Functions for RDF (1)

SPARQL supports also RDF-specific filter functions:

`BOUND(A)`	`true` if `A` is a bound variable
`isURI(A)`	`true` if `A` is a URI
`isBLANK(A)`	`true` if `A` is a blank node
`isLITERAL(A)`	`true` if `A` is an RDF literal
`STR(A)`	the lexical form (`xsd:string`) of RDF literals or URIs
`LANG(A)`	language tag of an RDF literal (`xsd:string`) or empty string if no language tag is given
`DATATYPE(A)`	datatyp URI of an RDF literal (`xsd:string` for untyped literals without language tag)

Filter Functions: Special Functions for RDF (2)

Additional RDF specific filter functions:

`sameTERM(A,B)`	`true`, if `A` and `B` are the same RDF terms
`langMATCHES(A,B)`	`true`, if the language tag of `A` matches the pattern `B`
`REGEX(A,B)`	`true`, if the string `A` matches the regular expression `B`

Example:

```
PREFIX ex: <http://example.org/>
SELECT ?book WHERE
  { ?book    ex:review  ?text .
    FILTER ( langMATCHES( LANG(?text), "de") )
  }
```

Filter Functions: Boolean Operators

Filter conditions can be connected using Boolean operators: `&&`, `||`, `!`

Partially expressible with graph patterns:

- Conjunction corresponds to multiple filters
- Disjunction corresponds to filter expressions specified in alternative (`UNION`) patterns

Agenda

1 Introduction and Motivation

2 Simple SPARQL Queries

3 Complex Graph Patterns

4 Filters

5 Solution Modifiers

6 Conclusions & Outlook

Why solution modifiers?

So far, we have only seen basic formatting options for the results:

- How can we only receive parts of the results?
- How can we order results?
- Can we immediately eliminate duplicate results?

$\rightsquigarrow$ Solution sequence modifiers

Sorting Results

Sorting is achieved with the keyword `ORDER BY`

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

Sorting Results

Sorting is achieved with the keyword `ORDER BY`

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

- Sorting as with comparison operators in filters
- Alphabetical sorting of URIs as strings
- Order between elements of different types:
  unbound variables $<$ blank nodes $<$ URIs $<$ RDF literals
- Not all possibilities defined by the specification

Sorting Results

Sorting is achieved with the keyword `ORDER BY`

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

- Sorting as with comparison operators in filters
- Alphabetical sorting of URIs as strings
- Order between elements of different types:
  unbound variables $<$ blank nodes $<$ URIs $<$ RDF literals
- Not all possibilities defined by the specification

Further possible options:

- `ORDER BY DESC(?price)`: descending
- `ORDER BY ASC(?price)`: ascending (default)
- `ORDER BY DESC(?price), ?titel`: hierarchical ordering criteria

LIMIT, OFFSET and DISTINCT

Limit the set of results:

- `LIMIT`: Maximal number of results
- `OFFSET`: Position of the first returned result
- `SELECT DISTINCT`: Removal of duplicate results

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

$\rightsquigarrow$ `LIMIT` and `OFFSET` only meaningful with `ORDER BY`!

Agenda

1 Introduction and Motivation

2 Simple SPARQL Queries

3 Complex Graph Patterns

4 Filters

5 Solution Modifiers

6 Conclusions & Outlook

Overview of the Presented SPARQL Features

Basic Structure
PREFIX
WHERE

Graph Patterns
Basic Graph Patterns
{...}
OPTIONAL
UNION

Filter
BOUND
isURI
isBLANK
isLITERAL
STR
LANG
DATATYPE
sameTERM
langMATCHES
REGEX

Modifiers
ORDER BY
LIMIT
OFFSET
DISTINCT

Output Formats
SELECT

Summary

- We have encountered the main SPARQL 1.0 features through examples
  - Basic structures (prefixes, patterns)
  - Simple and complex patterns (alternatives, optional parts, groups
  - Filters
  - Modifiers
- Semantics is defined via translation to the SPARQL algebra
- So far only informally introduced

Outlook

Open Questions

- How does the algebra translation work?
- How can we evaluate SPARQL algebra objects?
- What extensions does SPARQL 1.1 cover?
- How does the SPARQL protocol work?
- How can we query for implicit consequences that follow under RDF(S) or OWL semantics?
- How difficult is it to implement SPARQL (with entailment)?
- ...