



TECHNISCHE
UNIVERSITÄT
DRESDEN

FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

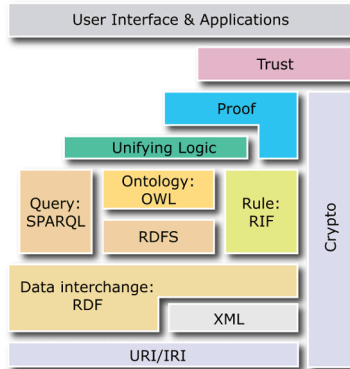
SPARQL Algebra

Sebastian Rudolph

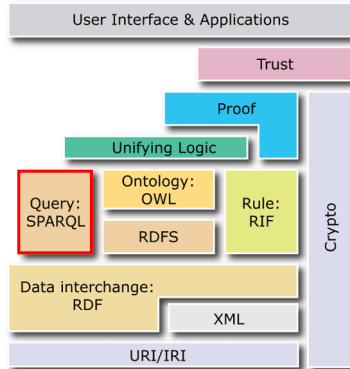


DRESDEN
concept
Dresden am
Wissenschaft
und Kultur

The SPARQL Query Language



The SPARQL Query Language



Agenda

- 1 Recap
- 2 Evaluation of the SPARQL Algebra
- 3 Operators for the Modifiers
- 4 Summary

Agenda

- 1 Recap
- 2 Evaluation of the SPARQL Algebra
- 3 Operators for the Modifiers
- 4 Summary

Recap: Introduced 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

CONSTRUCT

ASK

DESCRIBE

Translation into SPARQL Algebra

```
{ ?book ex:price ?price .  
  FILTER (?price < 15)  
  OPTIONAL { ?book ex:title ?title }  
  { ?book ex:author ex:Shakespeare } UNION  
  { ?book ex:author ex:Marlowe }  
}
```

Semantics of a SPARQL query:

- 1 Transformation of the query into an algebra expression
- 2 Evaluation of the algebra expression

Translation into SPARQL Algebra

```
{ ?book ex:price ?price
  FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
}
```

Attention: Filters apply to the whole group in which they occur

Translation into SPARQL Algebra

```
{ ?book ex:price ?price
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
  FILTER (?price < 15)
}
```

- 1 Expand abbreviated IRIs

Translation into SPARQL Algebra

```
{ ?book <http://ex.org/price> ?price
  OPTIONAL { ?book <http://ex.org/title> ?title }
  { ?book <http://ex.org/author>
      <http://ex.org/Shakespeare> } UNION
  { ?book <http://ex.org/author>
      <http://ex.org/Marlowe> }
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ ?book <http://ex.org/price> ?price
  OPTIONAL { ?book <http://ex.org/title> ?title }
  { ?book <http://ex.org/author>
      <http://ex.org/Shakespeare> } UNION
  { ?book <http://ex.org/author>
      <http://ex.org/Marlowe> }
  FILTER (?price < 15)
}
```

2. Replace triple patterns with operator $Bgp(\cdot)$

Translation into SPARQL Algebra

```
{ Bgp(?book <http://ex.org/price> ?price)
  OPTIONAL {Bgp(?book <http://ex.org/title> ?title)}
  {Bgp(?book <http://ex.org/author>
    <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
    <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ Bgp(?book <http://ex.org/price> ?price)
  OPTIONAL {Bgp(?book <http://ex.org/title> ?title)}
  {Bgp(?book <http://ex.org/author>
      <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
      <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

3. Introduce the LeftJoin(·) operator for optional parts

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
           Bgp(?book <http://ex.org/title> ?title),
           true)
  {Bgp(?book <http://ex.org/author>
       <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
       <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
           Bgp(?book <http://ex.org/title> ?title),
           true)
  {Bgp(?book <http://ex.org/author>
       <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
       <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

4. Combine alternative graph patterns with Union(.) operator
- ↪ Refers to neighbouring patterns and has higher precedence than conjunction (left associative)

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
          Bgp(?book <http://ex.org/title> ?title),  
          true)  
  Union(Bgp(?book <http://ex.org/author>  
        <http://ex.org/Shakespeare>),  
        Bgp(?book <http://ex.org/author>  
            <http://ex.org/Marlowe>))  
  FILTER (?price < 15)  
}
```


Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
          Bgp(?book <http://ex.org/title> ?title),  
          true)  
  Union(Bgp(?book <http://ex.org/author>  
        <http://ex.org/Shakespeare>),  
        Bgp(?book <http://ex.org/author>  
            <http://ex.org/Marlowe>))  
  FILTER (?price < 15)  
}
```

5. Apply Join(.) operator to join non-filter elements

Translation into SPARQL Algebra

```
{ Join(  
  LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
    Bgp(?book <http://ex.org/title> ?title),  
    true),  
  Union(Bgp(?book <http://ex.org/author>  
    <http://ex.org/Shakespeare>),  
    Bgp(?book <http://ex.org/author>  
    <http://ex.org/Marlowe>)))  
FILTER (?price < 15)  
}
```

Translation into SPARQL Algebra

```
{ Join(  
  LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
    Bgp(?book <http://ex.org/title> ?title),  
    true),  
  Union(Bgp(?book <http://ex.org/author>  
    <http://ex.org/Shakespeare>),  
    Bgp(?book <http://ex.org/author>  
    <http://ex.org/Marlowe>)))  
FILTER (?price < 15)  
}
```

6. Translate a group with filters with the Filter(-) operator

Translation into SPARQL Algebra

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
             Bgp(?book <http://ex.org/title> ?title),  
            true),  
    Union(Bgp(?book <http://ex.org/author>  
           <http://ex.org/Shakespeare>),  
          Bgp(?book <http://ex.org/author>  
              <http://ex.org/Marlowe>))))
```

Translation into SPARQL Algebra

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
      Bgp(?book <http://ex.org/title> ?title),  
      true),  
    Union(Bgp(?book <http://ex.org/author>  
      <http://ex.org/Shakespeare>),  
      Bgp(?book <http://ex.org/author>  
        <http://ex.org/Marlowe>))))
```

- Online translation tool:
<http://sparql.org/query-validator.html>

Agenda

- 1 Recap
- 2 Evaluation of the SPARQL Algebra
- 3 Operators for the Modifiers
- 4 Summary

Semantics of the SPARQL Algebra Operations

$\text{Bgp}(P)$	match/evaluate pattern P
$\text{Join}(M_1, M_2)$	conjunctive join of solutions M_1 and M_2
$\text{Union}(M_1, M_2)$	union of solutions M_1 with M_2
$\text{LeftJoin}(M_1, M_2, F)$	optional join of M_1 with M_2 with filter constraint F (<code>true</code> if no filter given)
$\text{Filter}(F, M)$	filter solutions M with constraint F
Z	empty pattern (identity for join)

SPARQL Solutions

Solutions as partial functions

- Domain: variables from the query
- Range: IRIs \cup blank nodes \cup RDF literals
- Assignment σ for blank nodes in the query
- Evaluation $\llbracket \text{Bgp}(P) \rrbracket_G$ of a BGP P over a graph G results in a multi set

Union of Solutions

Definition (Compatibility & Union)

Two solutions μ_1 and μ_2 are compatible if $\mu_1(x) = \mu_2(x)$ for all x , for which μ_1 and μ_2 are defined.
The union of two compatible solutions μ_1 and μ_2 is defined as:

$$(\mu_1 \cup \mu_2)(x) = \begin{cases} \mu_1(x) & \text{if } x \in \text{dom}(\mu_1) \\ \mu_2(x) & \text{otherwise} \end{cases}$$

↔ simple intuition: union of matching table rows

Union of Solutions

Definition (Compatibility & Union)

Two solutions μ_1 and μ_2 are compatible if

$\mu_1(x) = \mu_2(x)$ for all x , for which μ_1 and μ_2 are defined.

The union of two compatible solutions μ_1 and μ_2 is defined as:

$$(\mu_1 \cup \mu_2)(x) = \begin{cases} \mu_1(x) & \text{if } x \in \text{dom}(\mu_1) \\ \mu_2(x) & \text{otherwise} \end{cases}$$

↪ simple intuition: union of matching table rows

- We now also define the evaluation of the other SPARQL algebra operators

Evaluation of Join(\cdot)

For the evaluation of $\text{Join}(A_1, A_2)$ over a graph G with A_1, A_2 algebra objects, we define:

- Let $M_1 = \llbracket A_1 \rrbracket_G$
 - Let $M_2 = \llbracket A_2 \rrbracket_G$
 - Let $J(\mu) = \{(\mu_1, \mu_2) \mid M_1(\mu_1) > 0, M_2(\mu_2) > 0, \\ \mu_1 \text{ and } \mu_2 \text{ are compatible and } \mu = \mu_1 \cup \mu_2\}$
- $\rightsquigarrow J$ defines compatible pairs of solutions from M_1 and M_2

The evaluation $\llbracket \text{Join}(A_1, A_2) \rrbracket_G$ results in

$$\left\{ (\mu, n) \mid n = \sum_{(\mu_1, \mu_2) \in J(\mu)} (M_1(\mu_1) * M_2(\mu_2)) > 0 \right\}$$

Example to Join(\cdot)

We consider $\text{Join}(A_1, A_2)$ over a graph G with $\llbracket A_1 \rrbracket_G = M_1$, $\llbracket A_2 \rrbracket_G = M_2$ and:

$$M_1 = \{((\mu_1: ?x \mapsto \text{ex} : a, ?y \mapsto \text{ex} : b), 2),$$

$$((\mu_2: ?x \mapsto \text{ex} : a, 1))\}$$

$$M_2 = \{((\mu_3: ?y \mapsto \text{ex} : b, ?z \mapsto \text{ex} : c, 3))\}$$

$$\mu = ?x \mapsto \text{ex} : a, ?y \mapsto \text{ex} : b, ?z \mapsto \text{ex} : c$$

$$J(\mu) = \{(\mu_1, \mu_3), (\mu_2, \mu_3)\}$$

$$\text{Join}(M_1, M_2) = \left\{ (\mu, n) \mid n = \sum_{(\mu_1, \mu_2) \in J(\mu)} (M_1(\mu_1) * M_2(\mu_2)) > 0 \right\}$$

$$= \{(\mu, 9)\}$$

$$n = 2 * 3 + 1 * 3 = 6 + 3 = 9$$

Evaluation of Union(\cdot)

The evaluation of $\text{Union}(A_1, A_2)$ over a graph G , written $\llbracket \text{Union}(A_1, A_2) \rrbracket_G$, with A_1, A_2 algebra objects results in:

$$\left\{ (\mu, n) \mid M_1 = \llbracket A_1 \rrbracket_G, M_2 = \llbracket A_2 \rrbracket_G, n = M_1(\mu) + M_2(\mu) > 0 \right\}$$

Evaluation of Filter(\cdot)

The evaluation of $\text{Filter}(F, A)$ over a graph G , written $\llbracket \text{Filter}(F, A) \rrbracket_G$, with F a filter condition and A an algebra object results in:

$$\left\{ (\mu, n) \mid M = \llbracket A \rrbracket_G, M(\mu) = n > 0 \text{ and } \llbracket \mu(F) \rrbracket = \text{true} \right\}$$

$\llbracket \mu(F) \rrbracket$ is the Boolean result of evaluating the filter condition

Evaluation of LeftJoin(\cdot)

The evaluation of $\text{LeftJoin}(A_1, A_2, F)$ over a graph G with F a filter condition and A_1, A_2 algebra objects is defined as:

- $M_1 = \llbracket A_1 \rrbracket_G$
- $M_2 = \llbracket A_2 \rrbracket_G$

The evaluation $\llbracket \text{LeftJoin}(A_1, A_2, F) \rrbracket_G$ of $\text{LeftJoin}(A_1, A_2, F)$ over G results in

$$\llbracket \text{Filter}(F, \text{Join}(A_1, A_2)) \rrbracket_G \cup \left\{ (\mu_1, M_1(\mu_1)) \mid \text{for all } \mu_2 \text{ with } M_2(\mu_2) > 0 : \mu_1 \text{ and } \mu_2 \text{ are incompatible or } \llbracket (\mu_1 \cup \mu_2)(F) \rrbracket = \text{false} \right\}$$

Example

```
@prefix ex: <http://eg.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet          ex:author  ex:Shakespeare ;
                  ex:price   "10.50"^^xsd:decimal .
ex:Macbeth        ex:author  ex:Shakespeare .
ex:Tamburlaine    ex:author  ex:Marlowe ;
                  ex:price   "17"^^xsd:integer .
ex:DoctorFaustus ex:author  ex:Marlowe ;
                  ex:price   "12"^^xsd:integer ;
                  ex:title   "The Tragical History of Doctor Faustus" .
ex:RomeoJulia     ex:author  ex:Brooke ;
                  ex:price   "9"^^xsd:integer .
```

```
{ ?book ex:price ?price . FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title . }
  { ?book ex:author ex:Shakespeare . } UNION
  { ?book ex:author ex:Marlowe . }
}
```



```

@prefix ex: <http://eg.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet          ex:author  ex:Shakespeare ;
                   ex:price  "10.50"^^xsd:decimal .
ex:Macbeth         ex:author  ex:Shakespeare .
ex:Tamburlaine     ex:author  ex:Marlowe ;
                   ex:price  "17"^^xsd:integer .
ex:DoctorFaustus  ex:author  ex:Marlowe ;
                   ex:price  "12"^^xsd:integer ;
                   ex:title  "The Tragical History of Doctor Faustus" .
ex:RomeoJulia     ex:author  ex:Brooke ;
                   ex:price  "9"^^xsd:integer .

```

```

Filter(?price < 15,
  Join(LeftJoin(Bgp(?book <http://eg.org/price> ?price),
    Bgp(?book <http://eg.org/title> ?title), true),
    Union(Bgp(?book <http://eg.org/author>
      <http://eg.org/Shakespeare>),
      Bgp(?book <http://eg.org/author>
        <http://eg.org/Marlowe>))))

```

Example Evaluation (1)

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),  
      Bgp(?book <http://eg.org/title> ?title),  
      true),  
    Union(Bgp(?book <http://eg.org/author>  
      <http://eg.org/Shakespeare>),  
      Bgp(?book <http://eg.org/author>  
        <http://eg.org/Marlowe>))))))
```

book
ex:Tamburlaine
ex:DoctorFaustus

Example Evaluation (1)

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),  
             Bgp(?book <http://eg.org/title> ?title),  
            true),  
    Union(Bgp(?book <http://eg.org/author>  
           <http://eg.org/Shakespeare>),  
          Bgp(?book <http://eg.org/author>  
              <http://eg.org/Marlowe>))))
```

book
ex:Tamburlaine
ex:DoctorFaustus

book
ex:Macbeth
ex:Hamlet

Example Evaluation (2)

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),  
             Bgp(?book <http://eg.org/title> ?title),  
            true),  
    Union(Bgp(?book <http://eg.org/author>  
           <http://eg.org/Shakespeare>),  
          Bgp(?book <http://eg.org/author>  
              <http://eg.org/Marlowe>))))
```

book
ex:Hamlet
ex:Macbeth
ex:Tamburlaine
ex:DoctorFaustus

Example Evaluation (3)

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),  
      Bgp(?book <http://eg.org/title> ?title),  
      true),  
    Union(Bgp(?book <http://eg.org/author>  
      <http://eg.org/Shakespeare>),  
      Bgp(?book <http://eg.org/author>  
        <http://eg.org/Marlowe>))))
```

book	price
ex:Hamlet	10.5
ex:Tamburlaine	17
ex:DoctorFaustus	12
ex:RomeoJulia	9

Example Evaluation (3)

```
Filter(?price < 15,
  Join(
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),
      Bgp(?book <http://eg.org/title> ?title),
      true),
    Union(Bgp(?book <http://eg.org/author>
      <http://eg.org/Shakespeare>),
      Bgp(?book <http://eg.org/author>
        <http://eg.org/Marlowe>))))
```

book	price
ex:Hamlet	10.5
ex:Tamburlaine	17
ex:DoctorFaustus	12
ex:RomeoJulia	9

book	title
ex:DoctorFaustus	"The Tragical History of Doctor Faustus"

Example Evaluation (4)

```
Filter(?price < 15,
  Join(
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),
      Bgp(?book <http://eg.org/title> ?title),
      true),
    Union(Bgp(?book <http://eg.org/author>
      <http://eg.org/Shakespeare>),
      Bgp(?book <http://eg.org/author>
        <http://eg.org/Marlowe>))))
```

book	price	title
ex:Hamlet	10.5	
ex:Tamburlaine	17	
ex:DoctorFaustus	12	"The Tragical History of Doctor Faustus"
ex:RomeoJulia	9	

Example Evaluation (5)

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),  
      Bgp(?book <http://eg.org/title> ?title),  
      true),  
    Union(Bgp(?book <http://eg.org/author>  
      <http://eg.org/Shakespeare>),  
      Bgp(?book <http://eg.org/author>  
        <http://eg.org/Marlowe>))))
```

book	price	title
ex:Hamlet	10.5	
ex:Tamburlaine	17	
ex:DoctorFaustus	12	"The Tragical History of Doctor Faustus"

Example Evaluation (6)

```
Filter(?price < 15,  
  Join(  
    LeftJoin(Bgp(?book <http://eg.org/price> ?price),  
      Bgp(?book <http://eg.org/title> ?title),  
      true),  
    Union(Bgp(?book <http://eg.org/author>  
      <http://eg.org/Shakespeare>),  
      Bgp(?book <http://eg.org/author>  
        <http://eg.org/Marlowe>))))
```

book	price	title
ex:Hamlet	10.5	
ex:DoctorFaustus	12	"The Tragical History of Doctor Faustus"

Agenda

- 1 Recap
- 2 Evaluation of the SPARQL Algebra
- 3 Operators for the Modifiers**
- 4 Summary

Operators for Representing the Modifiers

$\text{ToList}(M)$	Constructs from a multi set a sequence with the same elements and multiplicity (arbitrary order, duplicates not necessarily adjacent)
$\text{OrderBy}(M, \text{comparators})$	sorts the solutions
$\text{Distinct}(M)$	removes the duplicates
$\text{Reduced}(M)$	may remove duplicates
$\text{Slice}(M, o, l)$	cuts the solutions to a list of length l starting from position o
$\text{Project}(M, \text{vars})$	projects out the mentioned variables

Transformation of the Modifiers

Let q be a SPARQL query with pattern P and corresponding algebra object A_P .
We construct an algebra object A_q for q as follows:

- 1 $A_q := \text{ToList}(A_P)$
- 2 $A_q := \text{OrderBy}(A_q, (c_1, \dots, c_n))$ if q contains an ORDER BY clause with comparators c_1, \dots, c_n
- 3 $A_q := \text{Project}(A_q, \text{vars})$ if the result format is SELECT with vars the selected variables (* all variables in scope)
- 4 $A_q := \text{Distinct}(A_q)$ if the result format is SELECT and q contains DISTINCT
- 5 $A_q := \text{Reduced}(A_q)$ if the result format is SELECT and q contains REDUCED
- 6 $A_q := \text{Slice}(A_q, \text{start}, \text{length})$ if the query contains OFFSET start or LIMIT length where start defaults to 0 and length defaults to $(\|A_q\|_G - \text{start})$

Evaluation of the Modifiers

The algebra objects for the modifiers are recursively evaluated

- Evaluate the algebra expression of the operator
- Apply the operations for the solution modifiers to the obtained solutions

Agenda

- 1 Recap
- 2 Evaluation of the SPARQL Algebra
- 3 Operators for the Modifiers
- 4 Summary**

Summary

- We learned how to evaluate SPARQL queries
- The query is transformed into an algebra object
- The query basic graph patterns generate solutions
- The other operators combine solutions
- The result format determines how the solutions are presented

Outlook

- Next lecture: SPARQL 1.1 features
- Non-Query parts of the specification (Protocol, Service Descriptions, Update, ...)
- Then: Entailment Regimes (SPARQL with inferred results)