RDF AND SPARQL

Part IV: Syntax of SPARQL

Sebastian Rudolph
ICCL Summer School

Dresden, August 2013
Agenda

1. Introduction and Motivation
2. Simple SPARQL Queries
3. Complex Graph Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. Conclusions & Outlook
Agenda

1. Introduction and Motivation
2. Simple SPARQL Queries
3. Complex Graph Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. 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 entailment does not cover all kinds of information needs

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

Expressible neither in RDF nor in OWL
Do OWL and RDF(S) not suffice?

Even OWL entailment does not cover all kinds of information needs

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

Expressible neither in RDF nor in OWL

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

1. Introduction and Motivation
2. Simple SPARQL Queries
3. Complex Graph Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. 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 }

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 URIs are possible (PREFIX).

Query result for the selected variables (SELECT).
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 URIs 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 URIs 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 <mailto:b.glimm@googlemail.com> ;
   foaf:icqChatID "b.glimm" ;
   foaf:aimChatID "b.glimm" .
_:b foaf:name "Sebastian Rudolph" ;
   foaf:mbox "sebastian.rudolph@tu-dresden.de" .
_: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 <mailto:b.glimm@googlemail.com> ;
    foaf:icqChatID "b.glimm" ;
    foaf:aimChatID "b.glimm" .
_:b foaf:name "Sebastian Rudolph" ;
    foaf:mbox "sebastian.rudolph@tu-dresden.de" .
_:c foaf:name "Pascal Hitzler" ;
    foaf:aimChatID "phi" .
foaf:icqChatID rdfs:subPropertyOf foaf:nick .
foaf:name rdfs:domain foaf:Person .

BGP matching results:

<table>
<thead>
<tr>
<th>x</th>
<th>name</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:a</td>
<td>&quot;Birte Glimm&quot;</td>
<td><a href="mailto:b.glimm@googlemail.com">mailto:b.glimm@googlemail.com</a></td>
</tr>
<tr>
<td>_:b</td>
<td>&quot;Sebastian Rudolph&quot;</td>
<td>&quot;<a href="mailto:sebastian.rudolph@tu-dresden.de">sebastian.rudolph@tu-dresden.de</a>&quot;</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>&quot;Birte Glimm&quot;</td>
<td><a href="mailto:b.glimm@googlemail.com">mailto:b.glimm@googlemail.com</a></td>
</tr>
<tr>
<td>b</td>
<td>&quot;Sebastian Rudolph&quot;</td>
<td>&quot;<a href="mailto:sebastian.rudolph@tu-dresden.de">sebastian.rudolph@tu-dresden.de</a>&quot;</td>
</tr>
</tbody>
</table>

Query results:

<table>
<thead>
<tr>
<th>name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Birte Glimm&quot;</td>
<td><a href="mailto:b.glimm@googlemail.com">mailto:b.glimm@googlemail.com</a></td>
</tr>
<tr>
<td>&quot;Sebastian Rudolph&quot;</td>
<td>&quot;<a href="mailto:sebastian.rudolph@tu-dresden.de">sebastian.rudolph@tu-dresden.de</a>&quot;</td>
</tr>
</tbody>
</table>
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 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:
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:

<table>
<thead>
<tr>
<th>subj</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:a</td>
<td>&quot;for&quot;</td>
</tr>
<tr>
<td>_:b</td>
<td>&quot;example&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>subj</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:y</td>
<td>&quot;for&quot;</td>
</tr>
<tr>
<td>_:g</td>
<td>&quot;example&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>subj</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:z</td>
<td>&quot;for&quot;</td>
</tr>
<tr>
<td>_:z</td>
<td>&quot;example&quot;</td>
</tr>
</tbody>
</table>
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**

- `/~` evaluation over the default graph

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

- `/~` evaluation over a named graph

**FROM clause**

- `/~` creation of a fresh default graph for the query
Example for Named Graphs

**Query with `FROM NAMED` clause**

```sparql
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" . }

$\Rightarrow$ ex:ex1 is the only result
$\Rightarrow$ Exact match for the datatypes is required

But: Abbreviations for numerical values allowed

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

$\Rightarrow$ 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 Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. Conclusions & Outlook
Group Graph Patterns

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

Example:

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

⇝ Only useful in combination with additional constructors
Optional Patterns

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

Example:

```sparql
  OPTIONAL { ?book ex:author ?author . }
}
```
Optional Patterns

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

Example:

```sparql
  OPTIONAL { ?book ex:titel ?titel . }
  OPTIONAL { ?book ex:author ?author . }
}
```

⇝ Parts of the query result can be unbound:

<table>
<thead>
<tr>
<th>book</th>
<th>titel</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://ex.org/book1">http://ex.org/book1</a></td>
<td>&quot;Titel1&quot;</td>
<td><a href="http://ex.org/author1">http://ex.org/author1</a></td>
</tr>
<tr>
<td><a href="http://ex.org/book2">http://ex.org/book2</a></td>
<td>&quot;Titel2&quot;</td>
<td></td>
</tr>
<tr>
<td><a href="http://ex.org/book3">http://ex.org/book3</a></td>
<td>&quot;Titel3&quot;</td>
<td>_:_a</td>
</tr>
<tr>
<td><a href="http://ex.org/book4">http://ex.org/book4</a></td>
<td></td>
<td>_:_a</td>
</tr>
<tr>
<td><a href="http://ex.org/book5">http://ex.org/book5</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Alternative Patterns

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

Example:

```
  { ?book ex:author ?author . } UNION
  { ?book ex:editor ?author . }
}
```

⇝ 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
Combination of Optional and Alternatives (1)

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

**Example**

```sparql
    {   ?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: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

```sparql
  { ?book ex:author ?author . } UNION
  { ?book ex:editor ?author . } OPTIONAL
  { ?author ex:surname ?name . } }
```

is equivalent to

Example with explicit grouping

```sparql
  { { ?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 Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. 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?”

Filter as a general mechanism for such expressions
Filter in SPARQL

Example:

PREFIX ex: <http://ex.org/>
SELECT ?book WHERE
    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
  ~> 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 )
SPARQL supports also **RDF-specific filter functions**:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND(A)</td>
<td>true if A is a bound variable</td>
</tr>
<tr>
<td>isURI(A)</td>
<td>true if A is a URI</td>
</tr>
<tr>
<td>isBLANK(A)</td>
<td>true if A is a blank node</td>
</tr>
<tr>
<td>isLITERAL(A)</td>
<td>true if A is an RDF literal</td>
</tr>
<tr>
<td>STR(A)</td>
<td>the lexical form (xsd:string) of RDF literals or URIs</td>
</tr>
<tr>
<td>LANG(A)</td>
<td>language tag of an RDF literal (xsd:string) or empty string if no language tag is given</td>
</tr>
<tr>
<td>DATATYPE(A)</td>
<td>datatype URI of an RDF literal (xsd:string for untyped literals without language tag)</td>
</tr>
</tbody>
</table>
Filter Functions: Special Functions for RDF (2)

Additional **RDF specific filter functions**:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sameTERM(A,B)</code></td>
<td>true, if A and B are the same RDF terms</td>
</tr>
<tr>
<td><code>langMATCHES(A,B)</code></td>
<td>true, if the language tag of A matches the pattern B</td>
</tr>
<tr>
<td><code>REGEX(A,B)</code></td>
<td>true, if the string A matches the regular expression B</td>
</tr>
</tbody>
</table>

Example:

```
PREFIX ex: <http://example.org/>
SELECT ?book WHERE
  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 Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. 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?

Solution sequence modifiers
Sorting is achieved with the keyword `ORDER BY`.

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

Sorting is achieved with the keyword `ORDER BY`

```sparql
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`

```sparql
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
ORDER BY ?price LIMIT 5 OFFSET 25
```

⇝ LIMIT and OFFSET only meaningful with ORDER BY!
Agenda

1. Introduction and Motivation
2. Simple SPARQL Queries
3. Complex Graph Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. Conclusions & Outlook
Output Format `SELECT` 

So far all results have been tables (solution sequences): Output format `SELECT` 

Syntax: `SELECT <VariableList>` or `SELECT *` 

**Advantage** 
Simple sequential processing of the results 

**Disadvantage** 
Structure/relationships between the objects in the results is lost
CONSTRUCT creates an RDF graph for the results

Example Query

```sparql
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person ex:mailbox ?email .  
               ?person ex:telephone ?tel . }
WHERE { ?person ex:email ?email .  
        ?person ex:tel ?tel . }
```
Output Format **CONSTRUCT**

**CONSTRUCT** creates an RDF graph for the results

**Example Query**

```sparql
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person ex:mailbox ?email .
  ?person ex:telephone ?tel . }
WHERE { ?person ex:email ?email .
  ?person ex:tel ?tel . }
```

**Advantage**

Structured result data with relationships between the elements

**Disadvantages**

- Sequential processing of the results is harder
- No treatment of unbound variables (triples are omitted)
CONSTRUCT Templates with Blank Nodes

Data

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:firstname "Alice" ;
      foaf:surname "Hacker" .
_:b foaf:firstname "Bob" ;
      foaf:surname  "Hacker" .
CONSTRUCT Templates with Blank Nodes

**Data**

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

**Query**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
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:surname ?fname
}
```
CONSTRUCT Templates with Blank Nodes

Resulting 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" .
Further Output Formats: **ASK & DESCRIBE**

SPARQL supports two additional output formats:
- **ASK** only checks whether the query has at least one answer (true/false result)
- **DESCRIBE** (informative) returns an RDF description for each resulting URI (application dependent)

**Example Query**

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

**Possible Result (prefix declarations omitted):**

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

foaf:mbox_sha1sum a owl:InverseFunctionalProperty .
```
Agenda

1. Introduction and Motivation
2. Simple SPARQL Queries
3. Complex Graph Pattern
4. Filters
5. Solution Modifiers
6. Output Formats
7. Conclusions & Outlook
Overview of the Presented SPARQL Features

<table>
<thead>
<tr>
<th>Basic Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFIX</td>
</tr>
<tr>
<td>WHERE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Graph Patterns</td>
</tr>
<tr>
<td>{...}</td>
</tr>
<tr>
<td>OPTIONAL</td>
</tr>
<tr>
<td>UNION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND</td>
</tr>
<tr>
<td>isURI</td>
</tr>
<tr>
<td>isBLANK</td>
</tr>
<tr>
<td>isLITERAL</td>
</tr>
<tr>
<td>STR</td>
</tr>
<tr>
<td>LANG</td>
</tr>
<tr>
<td>DATATYPE</td>
</tr>
<tr>
<td>sameTERM</td>
</tr>
<tr>
<td>langMATCHES</td>
</tr>
<tr>
<td>REGEX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER BY</td>
</tr>
<tr>
<td>LIMIT</td>
</tr>
<tr>
<td>OFFSET</td>
</tr>
<tr>
<td>DISTINCT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
</tr>
<tr>
<td>CONSTRUCT</td>
</tr>
<tr>
<td>ASK</td>
</tr>
<tr>
<td>DESCRIBE</td>
</tr>
</tbody>
</table>
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)?
• …