An Overview on
Linked Data Management
and SPARQL Querying

Olaf Hartig
http://olafhartig.de/foaf.rdf#olaf
@olafhartig

Database and Information Systems Research Group
Humboldt-Universität zu Berlin
Outline

Part 1  Introduction to SPARQL

Part 2  Storage Solutions for RDF Data

Part 3  Querying the Web of Linked Data
Outline

Part 1  Introduction to SPARQL

How do express declarative queries over RDF data?
(A user perspective)

Part 2

Part 3  Querying the Web of Linked Data
SPARQL in General

- A family of W3C recommendations

**SPARQL Query**
- Declarative query language for RDF data
- Our focus today

**SPARQL Update**
- Declarative update language for RDF data

**SPARQL Protocol**
- Communication between SPARQL processing services (a.k.a. SPARQL endpoints) and clients

**SPARQL Query Results XML Format**
- XML format for serializing query results
Main Idea of SPARQL Queries

- **Pattern matching:**
  - Describe subgraphs of the queried RDF graph
  - Subgraphs that match your description contribute an answer
  - Mean: graph patterns (i.e. RDF graphs with variables)
Main Idea of SPARQL Queries

Queried RDF graph:

```
?v

umbel-sc:Volcano

"1880"

p:lastEruption

dbpedia:Mount_Baker

dbpedia:Mount_Etna

rdf:type

umbel-sc:Volcano

dbpedia:Mount_Baker

dbpedia:Mount_Etna

Result:

?v

umbel-sc:Volcano

dbpedia:Mount_Baker

dbpedia:Mount_Etna
```
Components of SPARQL Queries

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>
SELECT ?v
FROM <http://example.org/myGeoData>
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
ORDER BY ?name
```

- **Prologue:**
  - **Prefix definitions** for using compact URIs (CURIEs)
  - **Attention** (difference to Turtle): No period (".") character as separator
Components of SPARQL Queries

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>
SELECT ?v
FROM <http://example.org/myGeoData>
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
ORDER BY ?name
```

• Result form specification:
  • SELECT, DESCRIBE, CONSTRUCT, or ASK
    (more about that later)
Components of SPARQL Queries

- Dataset specification:
  - Specify the RDF dataset to be queried
    (more about that later)
Components of SPARQL Queries

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>
SELECT ?v
FROM <http://example.org/myGeoData>
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
ORDER BY ?name
```

- **Query Pattern:**
  - WHERE clause specifies the graph pattern to be matched
Graph Patterns

• Different **types of graph patterns** for the query pattern (WHERE clause):
  • Basic graph pattern (BGP)
  • Group graph pattern
  • Optional graph pattern
  • Union graph pattern
  • Graph graph pattern
  • (Constraints)
Basic Graph Patterns

- Set of triple patterns (i.e. RDF triples with variables)
- Variable names prefixed with “?” or “$” (e.g. ?v, $v)
- Turtle syntax
  - Including syntactical sugar (e.g. property and object lists)

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>
SELECT ?name
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
```
Basic Graph Patterns

- Set of triple patterns (i.e. RDF triples with variables)
- Variable names prefixed with “?” or “$” (e.g. ?v, $v)
- Turtle syntax
  - Including syntactical sugar (e.g. property and object lists)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>

SELECT ?name
WHERE {
  ?v rdf:type umbel-sc:Volcano ;
    rdfs:label ?name .
}
```
Basic Graph Patterns (Example)

SELECT ?name WHERE {
  ?v rdf:type umbel-sc:Volcano;
  rdfs:label ?name .
}

*Prefix definitions omitted

Question: What are the names of all (known) volcanos?

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano;
  rdfs:label "Etna" .

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano.

dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
  umbel-sc:NaturalElevation;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .

*Prefix definitions omitted
Basic Graph Patterns (Example)

**Question:** What are the names of all (known) volcanos?

```
SELECT ?name WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label ?name . }
```

*Prefix definitions omitted*

**Data**

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano;
  rdfs:label "Etna" .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano. 
dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
  umbel-sc:NaturalElevation;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .
```

**Result:**

```
?name
"Etna"
"Бееренберг"@ru
"Beerenberg"@en
```
Basic Graph Patterns (Example)

Question: List all types of the volcano called “Beerenberg”

SELECT ?type WHERE {
  ?v rdf:type ?type ;
  rdfs:label "Beerenberg" .
}

Empty!
Basic Graph Patterns (Example)

- **Question:** List all types of the volcano called “Beerenberg”

```
SELECT ?type WHERE {
  ?v rdf:type ?type ;
  rdfs:label "Beerenberg"@en .
}
```

Data

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
    rdfs:label "Etna" .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano.
dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
    umbel-sc:NaturalElevation ;
    rdfs:label "Beerenberg"@en ;
    rdfs:label "Бееренберг"@ru .
```

Olaf Hartig - Linked Data Management and SPARQL Querying
Basic Graph Patterns (Example)

• Question: Where are all (known) volcanos located?
  (List the names of these locations)

• Blank nodes in SPARQL queries
  • As subject or object of a triple pattern
  • “Non-selectable” variables

```
SELECT ?name WHERE {
  _:x rdf:type umbel-sc:Volcano ;
  p:location dbpedia:United_States .
  dbpedia:United_States rdfs:label "United States" .
}
```

```
\text{Data}
\begin{align*}
  \text{dbpedia:Mount_Baker} & \text{ rdf:type } \text{umbel-sc:Volcano} ; \\
  & \text{ p:location } \text{dbpedia:United_States} . \\
  \text{dbpedia:United_States} & \text{ rdfs:label } \text{"United States"} .
\end{align*}
```

```
\text{Query}
\begin{align*}
  \text{SELECT } ?\text{name} \text{ WHERE } \{ \\
  & \_:\text{x} \text{ rdf:type } \text{umbel-sc:Volcano} ; \\
  & \text{ p:location } [ \text{ rdfs:label } ?\text{name} ] . \} \\
\end{align*}
```

?name
"United States"
Basic Graph Patterns (Example)

- Blank nodes in the queried data
  - Blank node identifiers may occur in the results

```
Data

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;
   p:location [ rdfs:label "United States"@en ,
                 "États-Unis"@fr ] .

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
   p:location [ rdfs:label "Italy" ] .
```

```
Query

SELECT ?l ?name WHERE {
   ?v rdf:type umbel-sc:Volcano ;
      p:location ?l .
}
```

```
?l ?name
---
_:x "United States"@en
_:x "États-Unis"@fr
_:y "Italy"
```
Optional Graph Patterns

```
SELECT ?v ?name WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label ?name .
}
```

- **Data**
  - `dbpedia:Beerenberg` rdf:type umbel-sc:Volcano ; rdfs:label "Beerenberg"@en .

- **Question:** What are all (known) volcanos and their names?

- **Problem:** Mount Baker **missing** (it has no name)

- **Table**
  - `dbpedia:Mount_Etna` | "Etna"
  - `dbpedia:Beerenberg` | "Beerenberg"@en
Optional Graph Patterns

- Keyword OPTIONAL allows for optional patterns

```
SELECT ?v ?name WHERE {
  ?v rdf:type umbel-sc:Volcano .
  OPTIONAL { ?v rdfs:label ?name }
}
```

<table>
<thead>
<tr>
<th>?v</th>
<th>?name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>&quot;Etna&quot;</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>&quot;Beerenberg&quot;@en</td>
</tr>
</tbody>
</table>

- Optional patterns may result in unbound variables
Union Graph Patterns

Data

\[
\begin{align*}
\text{dbpedia:Mount_Etna} & \text{ rdf:type umbel-sc:Volcano ; } \\
& \quad \text{rdfs:label "Etna" ;} \\
& \quad \text{p:location dbpedia:Italy .} \\
\text{dbpedia:Mount_Baker} & \text{ rdf:type umbel-sc:Volcano ;} \\
& \quad \text{p:location dbpedia:United_States .} \\
\text{dbpedia:Beerenberg} & \text{ rdf:type umbel-sc:Volcano ;} \\
& \quad \text{rdfs:label "Beerenberg"@en ;} \\
& \quad \text{p:location dbpedia:Norway .}
\end{align*}
\]

• Question: What volcanos are located in Italy or in Norway?

Query

\[
\text{SELECT ?v WHERE } \{ \\
?v \text{ rdf:type umbel-sc:Volcano ;} \\
\text{p:location ?} \}.
\]
Union Graph Patterns

- Union graph patterns allow for alternatives

```sparql
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  { ?v p:location dbp:Italy } 
  UNION 
  { ?v p:location dbp:Norway } 
}
```
Union Graph Patterns

- Union graph patterns allow for alternatives

```sparql
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  { ?v p:location dbpedia:Italy }
UNION
  { ?v p:location dbpedia:Norway }
}
```

Semantically equivalent

```sparql
SELECT ?v WHERE {
  { ?v rdf:type umbel-sc:Volcano ; p:location dbpedia:Italy }
UNION
  { ?v rdf:type umbel-sc:Volcano ; p:location dbpedia:Norway }
}
```
Group Graph Patterns

SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  { ?v p:location dbpedia:Italy } UNION
  { ?v p:location dbpedia:Norway }
}

Semantically equivalent

SELECT ?v WHERE {
  { ?v rdf:type umbel-sc:Volcano }
  { { ?v p:location dbpedia:Italy } UNION
    { ?v p:location dbpedia:Norway } }
}

Query
Constraints on Solutions

- Syntax: Keyword FILTER followed by filter expression

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/> 
PREFIX p: <http://dbpedia.org/property/>

SELECT ?v 
WHERE { 
    ?v rdf:type umbel-sc:Volcano; 
        p:lastEruption ?le . 
    FILTER ( ?le > 1900 ) 
}
```

- Filter expressions contain operators and functions
- Operators and functions operate on RDF terms
Constraints (Truth Table)

- Filter expressions evaluate to true, false, or error
- Truth table:

| A | B | A || B | A &&& B |
|---|---|------|--------|
| T | T |   T  |    T   |
| T | F |   T  |    F   |
| F | T |   T  |    F   |
| F | F |   F  |    F   |
| T | E |   T  |    E   |
| E | T |   T  |    E   |
| F | E |   E  |    F   |
| E | F |   E  |    F   |
| E | E |   E  |    E   |
Unary Operators in Constraints

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type(A)</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>! A</td>
<td>xsd:boolean</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>+ A</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>- A</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>BOUND(A)</td>
<td>variable</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>isURI(A)</td>
<td>RDF term</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>isBLANK(A)</td>
<td>RDF term</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>isLITERAL(A)</td>
<td>RDF term</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>STR(A)</td>
<td>literal / URI</td>
<td>simple literal</td>
</tr>
<tr>
<td>LANG(A)</td>
<td>literal</td>
<td>simple literal</td>
</tr>
<tr>
<td>DATATYPE(A)</td>
<td>literal</td>
<td>simple literal</td>
</tr>
</tbody>
</table>
Constraints (Example)

**Question:** List all types of the volcano called “Beerenberg”

```sparql
SELECT ?type WHERE {
  ?v rdf:type ?type ;
  rdfs:label ?name .
  FILTER ( STR(?name) = "Beerenberg" )
}
```

Data

```turtle
@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix umbel-sc: <http://umbel.sc/ontology/>

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
  umbel-sc:NaturalElevation ;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .
```
Constraints (Further Operators)

- **Binary operators:**
  - Logical connectives `&&` and `||` for `xsd:boolean`
  - Comparison operators `=`, `!=`, `<`, `>`, `<=`, and `>=` for numeric datatypes, `xsd:dateTime`, `xsd:string`, and `xsd:boolean`
  - Comparison operators `=` and `!=` for other datatypes
  - Arithmetic operators `+`, `-`, `*`, and `/` for numeric datatypes

- **Furthermore:**
  - REGEX(String,Pattern) or REGEX(String,Pattern,Flags)
  - sameTERM(A,B)
  - langMATCHES(A,B)
Constraints (Example)

SELECT ?v WHERE {
    ?v rdf:type umbel-sc:Volcano ;
    rdfs:label "Etna" .
}

SELECT ?v WHERE {
    ?v rdf:type umbel-sc:Volcano ;
    rdfs:label ?name .
    FILTER( REGEX(STR(?name),"e") )
}

• Question: What volcanos have an “e” in their name?
Constraints (Example)

- Question: What volcanos have an “e” in their name?

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label ?name .
  FILTER( REGEX(STR(?name),"e","i") )
}
```

Data

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
  umbel-sc:NaturalElevation ;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .
```
Graph Graph Patterns

- SPARQL queries are executed over an RDF dataset:
  - One default graph and
  - Zero or more named graphs (identified by an URI)
Components of SPARQL Queries

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>

SELECT ?v
FROM <http://example.org/myGeoData>
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
ORDER BY ?name
```

- **Dataset specification:**
  - Specify the RDF dataset to be queried
  (more about that later) here ...

- **Specification using FROM and FROM NAMED**
Graph Graph Patterns

- Evaluation of patterns w.r.t. **active graph**
- **GRAPH** clause for making a named graph the active graph
Graph Graph Patterns

```sparql
SELECT ?v WHERE {
  GRAPH <http://example.org/d1> {
    ?v rdf:type umbel-sc:Volcano .
  }
}
```

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .
}
```

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label "Beerenberg"@en .
}
```

```
dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
```

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .
```

```
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .
```

```
http://example.org/d2
```

```
http://example.org/d3
```

```
dbpedia:Mount_Etna http://example.org/d1
```

```
http://example.org/d2
```

Default Graph
Graph Graph Patterns

```
SELECT ?v WHERE {
    GRAPH ?g {
        ?v rdf:type umbel-sc:Volcano .
    }
}
```

Default Graph

```
dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
rdfs:label "Beerenberg" .
```

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
rdfs:label "Etna" .
```

```
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .
```

```
?v
```

```
http://example.org/d1
http://example.org/d2
http://example.org/d3
```
Graph Graph Patterns

dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.

```
SELECT ?v ?g
WHERE {
  GRAPH ?g {
    ?v rdf:type umbel-sc:Volcano ;
    rdfs:label "Etna" .
  }
}
```

```
SELECT ?v ?g
WHERE {
  GRAPH ?g {
    ?v rdf:type umbel-sc:Volcano .
  }
}
```

```
SELECT ?v ?g
WHERE {
  GRAPH ?g {
    ?v rdf:type umbel-sc:Volcano .
  }
}
```

Default Graph

http://example.org/d1
http://example.org/d2
http://example.org/d3
Graph Graph Patterns

SELECT ?v WHERE {
  _:x rdfs:seeAlso ?g
  GRAPH ?g {
    ?v rdf:type umbel-sc:Volcano .
  }
}

dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .
Negation

**Question:** What volcanos do not have a name in our data?

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  OPTIONAL { ?v rdfs:label ?name }
  FILTER( ! BOUND(?name) )
}
```

```
Premium: Mount Etna
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .
```

Negation as Failure
Negation

**Question:** What volcanos are **not** called “Beerenberg”?

```
SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  !?
  rdfs:label ?name .
  FILTER (STR(?name) != "Beerenberg")
}
```

Data

- **dbpedia:Mount_Etna** rdf:type umbel-sc:Volcano ;
  rdfs:label "Etna" .
- **dbpedia:Mount_Baker** rdf:type umbel-sc:Volcano .
- **dbpedia:Beerenberg** rdf:type umbel-sc:Volcano ;
  rdfs:label "Beerenberg"@en ;
  rdfs:label "Бееренберг"@ru .
Negation

SELECT ?v WHERE {
  ?v rdf:type umbel-sc:Volcano .
  OPTIONAL {
    FILTER (STR(?name) = "Beerenberg")
  }
  FILTER ( ! BOUND(?name) )
}

**Question:** What volcanos are *not* called “Beerenberg”?
Graph Graph Patterns

dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
rdfs:label "Etna" .

http://example.org/d1

dbpedia:Mount_Baker http://example.org/d2
  rdf:type umbel-sc:Volcano .

http://example.org/d2

dbpedia:Beerenberg http://example.org/d3
  rdf:type umbel-sc:Volcano ;
rdfs:label "Beerenberg"@en .

http://example.org/d3

• Question: Which named graphs contain the name of a volcano that is not referenced in the default graph?
Graph Graph Patterns

SELECT ?g WHERE {
  GRAPH ?g {
    ?v rdf:type umbel-sc:Volcano ;
    rdfs:label ?name .
  }
  OPTIONAL { ?v rdfs:seeAlso ?r }
  FILTER ( ! BOUND(?r) )
}

dbpedia:Mount_Etna rdfs:seeAlso <http://example.org/d1>.
dbpedia:Mount_Baker rdfs:seeAlso <http://example.org/d2>.

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
rdfs:label "Etna" .

http://example.org/d1

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .

http://example.org/d2

dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
rdfs:label "Beerenberg"@en .

http://example.org/d3

Question: Which named graphs contain the name of a volcano that is not referenced in the default graph?
Summary – Graph Patterns

• Different **types of graph patterns** for the query pattern (WHERE clause):
  • Basic graph pattern (BGP)
  • Group graph pattern
  • Optional graph pattern – keyword OPTIONAL
  • Union graph pattern – keyword UNION
  • Graph graph pattern – keyword GRAPH
  • Constraints – keyword FILTER
Components of SPARQL Queries

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>

SELECT ?v
FROM <http://example.org/myGeoData>
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
ORDER BY ?name
```

- Result form specification:
  - SELECT, DESCRIBE, CONSTRUCT, or ASK

  (more about that later) here ...
Result Forms

**SELECT**
- Result: sequence of solutions (i.e. sets of variable bindings)
- Selected variables separated by space (not by comma!)
- Asterisk character ("*") selects all variables in the pattern

**ASK**
- Check whether there is at least one result
- Result: true or false
- Example: Do we have data about volcanos?

```
ASK WHERE {
    ?v rdf:type umbel-sc:Volcano .
}
```
Result Forms

- **DESCRIBE**
  - Result: an RDF graph with data about resources
  - Non-deterministic (i.e. query processor defines the actual structure of the resulting RDF graph)
  - Example: just name the resource

```sparql
DESCRIBE <http://dbpedia.org/resource/Beerenberg>
```

- Example: Specify the resource(s) with a query pattern

```sparql
DESCRIBE ?v WHERE {
    ?v rdf:type umbel-sc:Volcano ;
    rdfs:label ?name .
}
```

- Multiple variables possible or asterisk (“*”) for all
RESULT FORMS

- **CONSTRUCT**
  - Result: an RDF graph constructed from a template
  - Template: graph pattern with variables from the query pattern

CONSTRUCT { ?v rdfs:label ?name ;
               rdf:type myTypes:VolcanosOutsideTheUS
}
WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label ?name .
  OPTIONAL { ?v p:location ?l
               FILTER ( ?l = dbpedia:United_States ) }
  FILTER ( ! BOUND(?l) )
}
Result Forms

```
dbpedia:Mount_Etna rdfs:label "Etna";
  rdf:type umbel-sc:Volcano;
  p:location dbpedia:Italy.

dbpedia:Mount_Baker rdfs:label "Mount Baker";
  rdf:type umbel-sc:Volcano;
  p:location dbpedia:United_States.

dbpedia:Beerenberg rdfs:label "Beerenberg" @en;
  rdf:type umbel-sc:Volcano;
  p:location dbpedia:Norway.
```

Data

```
dbpedia:Mount_Etna rdf:type myTypes:VolcanosOutsideTheUS.
dbpedia:Beerenberg rdf:type myTypes:VolcanosOutsideTheUS;
```

Result
Components of SPARQL Queries

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX umbel-sc: <http://umbel.org/umbel/sc/>
SELECT ?v
FROM <http://example.org/myGeoData>
WHERE {
  ?v rdf:type umbel-sc:Volcano .
}
ORDER BY ?name
```

- **Solution modifiers:**
  - Only for SELECT queries
  - Modify the `result set` as a whole (not single solutions)
  - Keywords: DISTINCT, ORDER BY, LIMIT, and OFFSET
Solution Modifiers

- **DISTINCT** removes duplicates from the result set

```sparql
SELECT ?type
WHERE { _:x rdf:type ?type }
```

```data
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
    rdfs:label "Etna" .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano.
dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
    umbel-sc:NaturalElevation ;
    rdfs:label "Beerenberg"@en ;
    rdfs:label "Бееренберг"@ru .
```

```data
DISTINCT
```

Additional Info:
- **umbel-sc:Volcano**
- **umbel-sc:NaturalElevation**
Solution Modifiers

- **DISTINCT** removes duplicates from the result set

```sparql
SELECT DISTINCT ?type
WHERE { _:x rdf:type ?type }
```

- Data

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
    rdfs:label "Etna" .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano.
dbpedia:Beerenberg rdf:type umbel-sc:Volcano,
    umbel-sc:NaturalElevation ;
    rdfs:label "Beerenberg"@en ;
    rdfs:label "Бееренберг"@ru .
```
Solution Modifiers

- **ORDER BY** orders the results

```
SELECT ?v WHERE { ?v rdf:type umbel-sc:Volcano ;
                  rdfs:label ?name } ORDER BY ?name
```

- How do we order different kinds of elements?
  unbound variable < blank node < URI < literal

- **ASC** for ascending (default) and **DESC** for descending

- Hierarchical order criteria:

```
SELECT ?name WHERE { ?v rdf:type umbel-sc:Volcano ;
                     p:lastEruption ?le ;
                     rdfs:label ?name } ORDER BY DESC(?le), ?name
```
Solution Modifiers

- **LIMIT** – limits the number of results

```
SELECT ?name WHERE { ?v rdf:type umbel-sc:Volcano ;
                      rdfs:label ?name }
ORDER BY ?name
LIMIT 5
```

- **OFFSET** – position/index of the first reported results

```
SELECT ?name WHERE { ?v rdf:type umbel-sc:Volcano ;
                      rdfs:label ?name }
ORDER BY ?name
LIMIT 5 OFFSET 10
```

- Order of result should be predictable
  (i.e. combine with ORDER BY)
SPARQL 1.1

• **New features of SPARQL 1.1 Query:**
  • Aggregate functions (e.g. COUNT, SUM, AVG)
  • Subqueries
  • Negation (EXISTS, NOT EXISTS, MINUS)
  • Assignments (e.g. BIND, SELECT expressions)
  • Property paths
  • Basic query federation (SERVICE, BINDINGS)

• **SPARQL 1.1 Update:**
  • Graph update (INSERT DATA, DELETE DATA, INSERT, DELETE, DELETE WHERE, LOAD, CLEAR)
  • Graph management (CREATE, DROP, COPY, MOVE, ADD)
Further Reading

- **W3C SPARQL Working Group**

- **J. Pérez, M. Arenas, and C. Gutierrez**: *Semantics and Complexity of SPARQL*. ACM Transactions on Database Systems 34(3), September 2009

- **SPARQL interface for DBpedia**: [http://dbpedia.org/snorql/](http://dbpedia.org/snorql/)
Outline

Part 1  Introduction to SPARQL

Part 2  Storage Solutions for RDF Data

Part 3  Querying the Web of Linked Data
Outline

Part 1  Introduction to SPARQL

Part 2  Storage Solutions for RDF Data

How can we manage RDF data in an efficient and scalable way?

• Use existing (relational) DBMS, or
• Create a new, native DBMS for RDF
Outline

Part 1  Introduction to SPARQL

Part 2  Storage Solutions for RDF Data
(RDF in a RDBMS)

Part 3  Querying the Web of Linked Data
Overview on RDF in RDBMS

- **Idea:** Use a specific relational schema for RDF data and benefit from 40 years of research in the DB community

- **3 steps for SPARQL query processing:**
  1. Convert SPARQL query to SQL query (w.r.t. the schema)
  2. Use RDBMS to answer the SQL query
  3. Generate SPARQL query result from the SQL query result

- **Relational schemas for RDF:**
  - Triple table
  - Property tables
  - Binary tables
  - Hybrid
**Triple Table (Basic Idea)**

- Store all RDF triples in a single table

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Triple Table (Basic Idea)

- **Store all RDF triples in a single table**

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>rdfs:label</td>
<td>&quot;Etna&quot;</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>rdfs:label</td>
<td>&quot;Beerenberg&quot;@en</td>
</tr>
</tbody>
</table>

- **Create indexes on combinations of S, P, and O**

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix umbel-sc: <http://umbelex.urhr.de/sc#> .

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
                rdfs:label "Etna" .

dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .

dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
                   rdfs:label "Beerenberg"@en .
```
Triple Table (Querying)

- Store all RDF triples in a single table

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>rdfs:label</td>
<td>&quot;Etna&quot;</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>rdfs:label</td>
<td>&quot;Beerenberg&quot;@en</td>
</tr>
</tbody>
</table>

SPARQL Query:
```sparql
SELECT ?name
WHERE {
  ?v rdf:type umbel-sc:Volcano ;
  rdfs:label ?name .
}
```

SQL Query:
```sql
SELECT t2.o AS name
FROM triples AS t1, triples AS t2
WHERE t1.p = 'rdf:type' AND
  t1.o = "umbel-sc:Volcano" AND
  t2.p = 'rdfs:label' AND
  t1.s = t2.s
```
### ID-based Triple Table

<table>
<thead>
<tr>
<th>RDF Term</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td></td>
</tr>
<tr>
<td>dbpedia:Mount_Etna</td>
<td></td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td></td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>rdfs:label</td>
<td>&quot;Etna&quot;</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>rdf:type</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>Beerenberg</td>
<td>rdfs:label</td>
<td>&quot;Beerenberg&quot;@en</td>
</tr>
</tbody>
</table>

- Use numerical identifier for each RDF term in the dataset
- Advantages:
  - Saves space
  - Enhances efficiency
## ID-based Triple Table

<table>
<thead>
<tr>
<th>RDF Term</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>1</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>2</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>3</td>
</tr>
<tr>
<td>rdf:type</td>
<td>4</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>5</td>
</tr>
<tr>
<td>umbel-sc:Volcano</td>
<td>6</td>
</tr>
<tr>
<td>&quot;Etna&quot;</td>
<td>7</td>
</tr>
<tr>
<td>&quot;Beerenberg&quot;@en</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

- Use numerical identifier for each RDF term in the dataset
- **Advantages:**
  - Saves space
  - Enhances efficiency
**Quad Table**

<table>
<thead>
<tr>
<th>G</th>
<th>S</th>
<th>P</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>102</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

- dbpedia:Mount_Etna: 1
- dbpedia:Mount_Baker: 2
- dbpedia:Beerenberg: 3
- rdf:type: 4
- rdfs:label: 5
- umbel-sc:Volcano: 6
- "Etna": 7
- "Beerenberg"@en: 8

- **Storing multiple RDF graphs**
- **Use Cases:**
  - Provenance
  - Versioning (multiple snapshots)
  - Contexts
Pros and Cons of Triple Tables

• **Pros:**
  • Easy to implement
  • If predicates are selective, not expensive

• **Cons:**
  • Many self joins are needed
  • If predicates are not selective, very expensive
Property Tables (Basic Idea)

- Combining all (or some) properties of similar subjects in $n$-ary tables

<table>
<thead>
<tr>
<th>S</th>
<th>rdf:type</th>
<th>rdfs:label</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>umbel-sc:Volcano</td>
<td>&quot;Etna&quot;</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>umbel-sc:Volcano</td>
<td>NULL</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>umbel-sc:Volcano</td>
<td>&quot;Beerenberg&quot;@en</td>
</tr>
</tbody>
</table>

- Also uses ID based encoding

```
@prefix dbpedia: <http://dbpedia.org/property/>
@prefix umbel-sc: <http://umbel.org/science/>

dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;
                 rdfs:label "Etna" .
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano .
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;
                 rdfs:label "Beerenberg"@en .
```
Pros and Cons of Property Tables

**Pros:**
- Fewer joins
- If the data is structured, we have a relational DB

**Cons:**
- Potentially a lot of NULLs
- Clustering is not trivial
- Multi-value properties are complicated
Binary Tables (Basic Idea)

- Also called *vertical partitioning*
- *n* two column tables  
  \( n \) is the number of unique properties in the data
- Also combined with ID based encoding of RDF terms

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Mount_Baker</td>
<td>umbel-sc:Volcano</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>umbel-sc:Volcano</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbpedia:Mount_Etna</td>
<td>&quot;Etna&quot;</td>
</tr>
<tr>
<td>dbpedia:Beerenberg</td>
<td>&quot;Beerenberg&quot;@en</td>
</tr>
</tbody>
</table>
Row Store vs. Column Store

- Different physical storage models for relational DBs

- **Row based storage:**
  - Tuples (i.e. DB records) are stored consecutively
  - Entire row needs to be read even if few attributes are projected

- **Column based storage:**
  - Read columns relevant to the query → projection is free
  - Inserts are expensive

- **Column based storage model fit binary tables very well**
Performance Comparison

Copied from:

However: property tables may outperform triple table and binary tables for RDF data with regular structure*

*J.J. Levandoski and M.F. Mokbel. *RDF data-centric storage*. ICWS 2009

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Geo. Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple Store</td>
<td>24.63</td>
<td>157</td>
<td>224.3</td>
<td>27.67</td>
<td>408.7</td>
<td>212.7</td>
<td>38.37</td>
<td>97</td>
</tr>
<tr>
<td>Prop. Table</td>
<td>12.66</td>
<td>18.37</td>
<td>579.8</td>
<td>28.54</td>
<td>47.85</td>
<td>101</td>
<td>6.1</td>
<td>38</td>
</tr>
<tr>
<td>Vert. Part.</td>
<td>12.66</td>
<td>41.7</td>
<td>71.3</td>
<td>35.49</td>
<td>52.34</td>
<td>84.6</td>
<td>13.25</td>
<td>36</td>
</tr>
<tr>
<td>C-Store</td>
<td>0.66</td>
<td>1.64</td>
<td>9.28</td>
<td>2.24</td>
<td>15.88</td>
<td>10.81</td>
<td>1.44</td>
<td>3</td>
</tr>
</tbody>
</table>
Pros and Cons of Binary Tables

• **Pros:**
  - Supports multi-value properties
  - No NULLs
  - Read only needed attributes (i.e. less I/O)
  - No clustering
  - Excellent performance (if number of properties is small, queries with bounded properties)

• **Cons:**
  - Expensive inserts
  - Bad performance (large number of properties, queries with unbounded properties)
Summary (RDF in RDBMS)

- Relational schemas for RDF:
  - Triple table
  - Property tables
  - Binary tables
  - Hybrid

- None of these approaches can compete with a purely relational model:

  “... there is still room for optimization in the proposed generic relational RDF storage schemes and thus new techniques for storing and querying RDF data are still required ...”

*S. Sakr and G. Al-Naymat. Relational Processing of RDF Queries: A Survey. SIGMOD Record 38(4), 2009*
Outline

Part 1  Introduction to SPARQL

Part 2  Storage Solutions for RDF Data

Part 3  Querying the Web of Linked Data
Interesting …
but, that was only about (local) RDF data.

What's about Linked Data on the Web?
Can we query the Web of Linked Data as if it were a single (gigantic) database?

SELECT DISTINCT ?i ?label
WHERE {
  ?prof rdf:type <http://res ... data/dbprofs#DBProfessor> ;
  foaf:topic_interest ?i .
  OPTIONAL {
    ?i rdfs:label ?label
    FILTER (LANG(?label)="en" || LANG(?label)=")
  }
} ORDER BY ?label
Traditional approach 1: **Data centralization** (e.g. data warehouse)

- Querying a collection of copies from all relevant datasets

  + Most efficient
  - Misses unknown or new sources
  - Collection possibly out of date
Query Federation Approach

Traditional approach 2: Federated query processing

- **Mediator** distributes subqueries to relevant sources and integrates the results

  + Current data
  - Sources must provide a query service
  - Misses unknown or new sources
Main Drawback

You have to know the relevant data sources in advance.

You restrict yourself to the selected sources.

You do not tap the full potential of the Web!
Link Traversal Based Query Execution*

Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - **Evaluate parts** of the query (triple patterns) on a continuously augmented set of data
  - **Look up URIs** in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links
- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset

Query

http://.../movie2449

actor_in

filmingLocation

?actor

lives_in

?loc

Queried data
Main Idea

- Intertwine query evaluation with traversal of data links
- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links
- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links
- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links
- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links
- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
Main Idea

- Intertwine query evaluation with traversal of data links

- We alternate between:
  - Evaluate parts of the query (triple patterns) on a continuously augmented set of data
  - Look up URIs in intermediate solutions and add retrieved data to the query-local dataset
## Summary

<table>
<thead>
<tr>
<th>data warehousing</th>
<th>search engines</th>
<th>query federation</th>
<th>active discovery query federation</th>
<th>link traversal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universe of Discourse (UoD): loaded data</td>
<td>Web of Data</td>
<td>known data sources</td>
<td>Web of Data</td>
<td>Web of Data</td>
</tr>
<tr>
<td>Required source interfaces: mainly RDF dumps</td>
<td>arbitrary</td>
<td>SPARQL endpoints</td>
<td>SPARQL endpoints</td>
<td>Linked Data (look-up) interface</td>
</tr>
<tr>
<td>Access to original data: no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Supporting data structures: indices and statistics</td>
<td>crawled index</td>
<td>statistics</td>
<td>(statistics)</td>
<td>-</td>
</tr>
<tr>
<td>Response and throughput: fast / fast</td>
<td>fast / fast</td>
<td>slow / medium</td>
<td>slow / slow</td>
<td>medium / slow</td>
</tr>
<tr>
<td>Recall (w.r.t. UoD): 100%</td>
<td>&lt;100%</td>
<td>100%</td>
<td>&lt;100%</td>
<td>&lt;100%</td>
</tr>
<tr>
<td>Precision: 100%</td>
<td>&lt;100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Up-to-dateness: low</td>
<td>medium</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
</tbody>
</table>

O. Hartig and A. Langegger. *A Database Perspective on Consuming Linked Data on the Web*. Datenbankspektrum 10(2), 2010
Outline

Part 1  Introduction to SPARQL

Part 2  Storage Solutions for RDF Data

Part 3  Querying the Web of Linked Data

Thanks!
These slides have been created by
Olaf Hartig

http://olafhartig.de

Many thanks to Juan Sequeda who provided most of the slides in the part about storage solutions for RDF data.

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 License (http://creativecommons.org/licenses/by-sa/3.0/)