Introduction to SPARQL
Acknowledgements

• This presentation is based on the W3C Candidate Recommendation “SPARQL Query Language for RDF” from http://www.w3.org/TR/rdf-sparql-query/

• Some of the material in this presentation is verbatim from the above Web site.
Presentation Outline

• Query languages for RDF and RDFS
• SPARQL: A Query Language for RDF
• Semantics of SPARQL
Query Languages for RDF and RDFS

- There have been many proposals for RDF and RDFS query languages:
  - RDQL (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/)
  - ICS-FORTH RQL (http://139.91.183.30:9090/RDF/RQL/) and SeRQL (http://www.openrdf.org/doc/sesame/users/ch06.html)
  - SPARQL (http://www.w3.org/TR/rdf-sparql-query/)
  - ...

In this course we will only cover SPARQL which is the current W3C recommendation for querying RDF data.
SPARQL

- SPARQL stands for “SPARQL Protocol and RDF Query Language”.

- In addition to the language, W3C has also defined:
  - The **SPARQL Protocol** for RDF specification: it defines the remote protocol for issuing SPARQL queries and receiving the results.
  - The **SPARQL Query Results** XML Format specification: it defines an XML document format for representing the results of SPARQL queries.
SPARQL 1.1

• In this lecture we will cover the SPARQL standard as of 2008.

• The standardization of SPARQL is carried out under the auspices of the W3C by the SPARQL working group.

• More information about ongoing work by this working group can be found at
  – http://www.w3.org/2009/sparql/wiki/Main_Page

• See http://www.w3.org/TR/sparql11-query/ for the new version of the SPARQL language (SPARQL 1.1).
SPARQL Basics

• SPARQL is based on matching graph patterns against RDF graphs.

• What is a graph pattern?

• To define graph patterns, we must first define triple patterns:
  – A **triple pattern** is like an RDF triple, but with the option of a variable in place of RDF terms (i.e., IRIs, literals or blank nodes) in the subject, predicate or object positions.

  – Example:

    ```
    ```

    – ?title is a variable.
SPARQL Graph Patterns

• We can distinguish the following kinds of graph patterns:
  – **Group** graph patterns. These are the more general case of graph pattern. They are build out of:
    • **Basic** graph patterns
    • **Filter** conditions
    • **Optional** graph patterns
    • **Alternative** graph patterns
  – Patterns on **named graphs**
Basic Graph Patterns

- A basic graph pattern (BGP) is a set of triple patterns written as a sequence of triple patterns (separated by a period if necessary).
- A BGP should be understood as the conjunction of its triple patterns.

Example:

```
?x foaf:name ?name . ?x foaf:mbox ?mbox
```
Group Graph Patterns

• A group graph pattern is a set of graph patterns delimited with braces { }.

• Simple examples:

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

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

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

• The above group graph patterns are equivalent. In general:
  – When a group graph pattern consists only of triple patterns or only of BGPs, these patterns are interpreted conjunctively, and the group graph pattern is equivalent to the corresponding set of triple patterns.
Group Graph Patterns (cont’d)

• {} is the **empty** group graph pattern.

• Group graph patterns are the **most general** kind of graph patterns; they can involve **other constructs** to be defined below. These constructs are introduced by certain **keywords**.

• **Important**: There is **no keyword for conjunction** (e.g., AND) in SPARQL. Conjunctive triple patterns or BGPs are simply juxtaposed and then enclosed in { and } to form a group graph pattern.
A Simple SPARQL Query

• Data:
  <http://example.org/book/book1>
  <http://purl.org/dc/elements/1.1/title>
  "SPARQL Tutorial".

• Query:
  SELECT ?title
  WHERE { <http://example.org/book/book1>
           <http://purl.org/dc/elements/1.1/title> ?title . }

• Result:

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SPARQL Tutorial&quot;</td>
</tr>
</tbody>
</table>
Comments

• Data will be presented using **Turtle**. The Turtle syntax is also utilized in SPARQL so it is useful to know it well.
• SELECT and WHERE clauses are like in SQL. But be careful: SPARQL and SQL are very different languages in general.
• Variables are like in Prolog or Datalog.
• Variables can also be written as $x$ instead of ?x.
• We can write `SELECT *` **like in SQL**.
• The result of a query is a set of **bindings** for the variables appearing in the SELECT clause. Bindings will be shown in tabular form.
Another Example

• Data:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .

_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .

_:c foaf:mbox <mailto:carol@example.org> .
Example (cont’d)

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

• **Result:**

<table>
<thead>
<tr>
<th>name</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Peter Goodguy&quot;</td>
<td><a href="mailto:peter@example.org">mailto:peter@example.org</a></td>
</tr>
<tr>
<td>&quot;Johnny Lee Outlaw&quot;</td>
<td><a href="mailto:jlow@example.com">mailto:jlow@example.com</a></td>
</tr>
</tbody>
</table>
Queries with RDF Literals

- We have to be careful when matching RDF literals (see the SPARQL specification for all the details). For example:

- Data:

```sparql
@prefix dt: <http://example.org/datatype#> .
@prefix ns: <http://example.org/ns#> .
@prefix : <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:x ns:p "cat"@en .
:y ns:p "42"^^xsd:integer .
:z ns:p "abc"^^dt:specialDatatype .
```
Matching RDF Literals (cont’d)

• The queries

```
SELECT ?v WHERE { ?v ?p "cat" }
```

and

```
SELECT ?v WHERE { ?v ?p "cat"@en }
```

have different results.

• Only the second one finds a matching triple and returns:

```
   v
```

```
<http://example.org/ns#x>
```
Blank Nodes in Query Results

• Data:
  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
  _:a foaf:name "Alice" .
  _:b foaf:name "Bob" .

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

• Result:

<table>
<thead>
<tr>
<th>x</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:c</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>_:d</td>
<td>&quot;Bob&quot;</td>
</tr>
</tbody>
</table>
Blank Nodes in Query Results (cont’d)

• Data:
  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
  _:a foaf:name "Alice" .
  _:b foaf:name "Bob" .
  _:a foaf:knows _:b .
  _:b foaf:knows _:a .

• Query:
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?x ?name1 ?y ?name2
  ?x foaf:knows ?y }

• Result:

<table>
<thead>
<tr>
<th>?x</th>
<th>name1</th>
<th>?y</th>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>_:c</td>
<td>&quot;Alice&quot;</td>
<td>_:d</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>_:d</td>
<td>&quot;Bob&quot;</td>
<td>_:c</td>
<td>&quot;Alice&quot;</td>
</tr>
</tbody>
</table>
Comments

• SPARQL does not consider blank nodes to be something like existentially quantified variables in FOL as semantics of RDF do!

• SPARQL considers blank nodes to be distinct constants scoped to the graph where they appear.

• **Example:** If we ask in the previous graph “How many resources with a name do we have?”, the answer is 2.

• See the paper
  Available from [http://axel.deri.ie/publications.html](http://axel.deri.ie/publications.html) for a comprehensive discussion of issues relating to blank nodes in the theory and practice of RDF and SPARQL.
Blank Nodes in Graph Patterns

• Blank nodes in graph patterns act as **variables**, not as references to specific blank nodes in the data being queried.

• Blank nodes **cannot appear** in a SELECT clause.

• The **scope** of blank node is the BGP in which it appears. A blank node which **appears more than once** in the same BGP stands for the same RDF term.

• The same blank node is **not** allowed to appear in two BGPs of the same query.

• **Important**: there is no reason to use blank nodes in a query; you can get the same functionality using variables.
Example

• **Data:**

```dot
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

• **Query:**

```dot
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { _:z foaf:name ?name . }
```

• **Result:**

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>“Bob”</td>
</tr>
</tbody>
</table>
Example (cont’d)

• **Data:**

```verbatim
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:b foaf:name "Bob".
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

• **Query:**

```verbatim
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
WHERE { _:z foaf:name ?name1 . _:v foaf:name ?name2 .
    _:z foaf:knows _:v }
```

• **Result:**

<table>
<thead>
<tr>
<th>name1</th>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>“Bob”</td>
<td>“Alice”</td>
</tr>
</tbody>
</table>
Example (cont’d)

- **Data:**
  ```
  @prefix foaf: <http://xmlns.com/foaf/0.1/> .

  _:a foaf:name "Alice" .
  _:b foaf:name "Bob" .
  _:a foaf:knows _:b .
  _:b foaf:knows _:a .
  ```

- **Query:**
  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?name1 ?name2
  WHERE { {_:z foaf:name ?name1} {_:z foaf:name ?name2} }
  ```

- **Result:** Error (blank node reused across basic graph patterns).
Query Forms

- The SELECT query form returns **variable bindings**.

- The CONSTRUCT query form returns an **RDF graph** specified by a graph template.

- The ASK query form can be used to **test whether or not a graph pattern has a solution**. No information is returned about the possible query solutions, just whether or not a solution exists.

- There is also a DESCRIBE query form which is not important and SPARQL does not prescribe any semantics for it.
Example - CONSTRUCT

• Data:

@prefix org: <http://example.com/ns#> .

_:a org:employeeName "Alice" .
_:a org:employeeId 12345 .
_:b org:employeeName "Bob" .
_:b org:employeeId 67890 .

• Query:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX org: <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name } WHERE { ?x org:employeeName ?name }
Example (cont’d)

• The result now is a graph:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
_:c foaf:name "Alice" .
_:d foaf:name "Bob" .
```
Examples - ASK

• Data:
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
_:a foaf:name "Alice" .  
_:a foaf:homepage <http://work.example.org/alice/> .  

_:b foaf:name "Bob" .  
_:b foaf:mbox <mailto:bob@work.example> .  

• Query:
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
ASK { ?x foaf:name "Alice" }  

• Answer:
  yes
Examples (cont’d)

• **Query:**
  ```sparql
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  ASK { ?x foaf:name "Alice" ;
       foaf:mbox <mailto:alice@work.example> }
  ```

• **Answer:**
  
  no

• **Note:** The answer should be understood as saying: **I couldn’t find bindings** to compute a solution to the given graph pattern.
Constraints on Variables

• The FILTER construct restricts variable bindings to those for which the filter expression evaluates to TRUE.
Example: Arithmetic Filters

• **Data:**

```sparql
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .

:book1 ns:price 42 .

```
Example (cont’d)

• Query:

```sql
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
    FILTER (?price < 30.5)
    ?x dc:title ?title . }
```

• Result:

<table>
<thead>
<tr>
<th>title</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;The Semantic Web&quot;</td>
<td>23</td>
</tr>
</tbody>
</table>
Example: String Filters

• **Query:**

```sparql
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE {
  ?x dc:title ?title
  FILTER regex(?title, "^SPARQL")
}
```

• **Result:**

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SPARQL Tutorial&quot;</td>
</tr>
</tbody>
</table>
Scope of Filters

• **Group graph patterns** are used to restrict the scope of FILTER conditions.

• A FILTER condition is a restriction on solutions over the whole group in which the filter appears.
Example

• The following graph patterns all have the same set of solutions:

- { ?x foaf:name ?name .
  FILTER regex(?name, "Smith") }  

- { FILTER regex(?name, "Smith")
  ?x foaf:name ?name .
  ?x foaf:mbox ?mbox . }  

- { ?x foaf:name ?name .
  FILTER regex(?name, "Smith")
  ?x foaf:mbox ?mbox . }
Comments

• We can have **multiple FILTERs** in a group graph pattern. They are equivalent to a single filter with conjoined filter conditions.

• FILTERs can be very complex Boolean conditions (see the SPARQL specification for details [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/) ).

• The regular expression language used by `regex` is defined in XQuery 1.0 and XPath 2.0.
Optional Graph Patterns

• Regular, complete structures cannot be assumed in all RDF graphs.

• It is useful to have queries that allow information to be added to the answer where the information is available, but do not reject the answer because some part of the query pattern does not match.

• Optional graph pattern matching provides this facility: if the optional part does not match, it creates no bindings but does not eliminate the solution.
Example

• Data:

```turtle
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.com> .
_:a foaf:mbox <mailto:alice@work.example> .

_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .
```
Example (cont’d)

• Query:

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

• Result:

<table>
<thead>
<tr>
<th>name</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td><a href="mailto:alice@example.com">mailto:alice@example.com</a></td>
</tr>
<tr>
<td>&quot;Alice&quot;</td>
<td><a href="mailto:alice@work.example">mailto:alice@work.example</a></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Semantics of Answers

• We can now see that answers to a SPARQL query can be formalized as sets of mappings i.e., partial functions from the set of variables to the set of RDF terms (URIs, literals and blank nodes).

• Later on we will give a complete formal semantics of SPARQL queries.
Example

• The answer of the previous query can be formalized by the following set of mappings:

```plaintext
{  
  { ?name → “Alice”, ?mbox → <mailto:alice@example.com> }  
  { ?name → “Alice”, ?mbox → <mailto:alice@work.example> }  
  { ?name → “Bob” }  
}
```
Optional Graph Patterns (cont’d)

• **Optional parts** of a graph pattern that we are trying to compute may be specified by starting with a graph pattern $P_1$ and then applying the **keyword** `OPTIONAL` to another graph pattern $P_2$ that follows it:

$$P_1 \text{ OPTIONAL } \{ \ P_2 \ \}$$
Properties of OPTIONAL

- OPTIONAL is a binary operator.

- OPTIONAL is left-associative:

  \[ P_1 \text{ OPTIONAL } \{ P_2 \} \text{ OPTIONAL } \{ P_3 \} \]

  is equivalent to

  \[ \{ P_1 \text{ OPTIONAL } \{ P_2 \} \} \text{ OPTIONAL } \{ P_3 \} \]
Properties of OPTIONAL (cont’d)

• The syntactic form
  \[
  \{ \text{OPTIONAL} \{ P \} \}\]
  is equivalent to
  \[
  \{ \{ \} \text{OPTIONAL} \{ P \} \}.
  \]

• In general
  \[
  \{ P1 \text{OPTIONAL} P2 \} \text{OPTIONAL} P3
  \]
  is not equivalent to
  \[
  P1 \text{OPTIONAL} \{ P2 \text{OPTIONAL} P3 \}.
  \]
FILTERs in Optional Pattern Matching

• The group graph pattern following a keyword OPTIONAL can of course be as complex as possible e.g., it can contain a FILTER.
Example

- **Data:**

  ```ttl
  @prefix dc: <http://purl.org/dc/elements/1.1/> .
  @prefix : <http://example.org/book/> .
  @prefix ns: <http://example.org/ns#> .


  :book2 ns:price 42 .

  ```
Example (cont’d)

- **Query:**

  ```sparql
  PREFIX dc: <http://purl.org/dc/elements/1.1/>
  PREFIX ns: <http://example.org/ns#>
  SELECT ?title ?price
    OPTIONAL { ?x ns:price ?price .
      FILTER (?price < 30) }
  }
  ```

- **Result:**

<table>
<thead>
<tr>
<th>title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>“SPARQL Tutorial”</td>
<td></td>
</tr>
<tr>
<td>“A New SPARQL Tutorial”</td>
<td></td>
</tr>
<tr>
<td>&quot;The Semantic Web&quot;</td>
<td>23</td>
</tr>
</tbody>
</table>
Notes that the OPTIONAL pattern in the previous query does not generate bindings in the following two cases:

- There is no `ns:price` property for ?x (e.g., when `?x=book1`).
- There is an `ns:price` property for ?x but its value is greater than or equal to 30 (e.g., when `?x=book2`).
Example with Multiple OPTIONALs

• Data:

```turtle
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:homepage <http://work.example.org/alice/> .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
```
Example (cont’d)

• Query:
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?name ?mbox ?hpage
  WHERE {
    ?x foaf:name ?name .
    OPTIONAL { ?x foaf:mbox ?mbox . }
    OPTIONAL { ?x foaf:homepage ?hpage . }
  }

• Result:

<table>
<thead>
<tr>
<th>name</th>
<th>mbox</th>
<th>hpage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td></td>
<td><a href="http://work.example.org/alice/">http://work.example.org/alice/</a></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td><a href="mailto:bob@work.example">mailto:bob@work.example</a></td>
<td></td>
</tr>
</tbody>
</table>
Properties of OPTIONAL (cont’d)

- The operator OPTIONAL has higher precedence than conjunction (remember: conjunction is encoded as juxtaposition of graph patterns).
Example

• Data

@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .


ex:book3 ns:price 34 .

Example (cont’d)

• Query 1:
  PREFIX ex: <http://example.org/>
  PREFIX dc: <http://purl.org/dc/elements/1.1/>
  PREFIX ns: <http://example.org/ns#>
  SELECT ?book ?title
            { ?book ns:price ?price .}
  }

• Answer:

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/book3">http://example.org/book3</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://example.org/book1">http://example.org/book1</a></td>
<td>&quot;Semantic Web&quot;</td>
</tr>
</tbody>
</table>
Example (cont’d)

• Give the precedence and associativity of OPTIONAL and conjunction, Query 1 is equivalent to the following query:

```sparql
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title
WHERE  
{  
    OPTIONAL { ?book dc:title ?title .} }
  
{ ?book ns:price ?price .}
}
```

• It is interesting to also see Query 2 below which has results different than Query 1.
Example (cont’d)

- **Query 2:**
  ```
  PREFIX ex: <http://example.org/>
  PREFIX dc: <http://purl.org/dc/elements/1.1/>
  PREFIX ns: <http://example.org/ns#>
  SELECT ?book ?title
  OPTIONAL { { ?book dc:title ?title .}
  { ?book ns:price ?price .} } }
  ```

- **Answer:**
<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/book3">http://example.org/book3</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://example.org/book2">http://example.org/book2</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://example.org/book1">http://example.org/book1</a></td>
<td>&quot;Semantic Web&quot;</td>
</tr>
</tbody>
</table>
Alternative Patterns (Disjunction)

• SPARQL provides a means of forming the disjunction of graph patterns so that one of several alternative graph patterns may match. If more than one of the alternatives match, all the possible pattern solutions are found.

• Pattern alternatives are syntactically specified with the keyword UNION.
Example

• Data:

```turtle
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .

_:a dc10:title "SPARQL Query Language Tutorial" .
_:a dc10:creator "Alice" .

_:b dc11:creator "Bob" .

_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL (updated)" .
```
Example (cont’d)

• Query:

```sql
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
```

• Result:

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SPARQL Protocol Tutorial&quot;</td>
</tr>
<tr>
<td>&quot;SPARQL&quot;</td>
</tr>
<tr>
<td>&quot;SPARQL (updated)&quot;</td>
</tr>
<tr>
<td>&quot;SPARQL Query Language Tutorial&quot;</td>
</tr>
</tbody>
</table>
Example (cont’d)

• Query:

```sql
PREFIX dc10: <http://purl.org/dc/elements/1.0/>  
PREFIX dc11: <http://purl.org/dc/elements/1.1/>  

SELECT ?author ?title  
WHERE  
{  
  UNION  
}
```

• Result:

<table>
<thead>
<tr>
<th>author</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>“SPARQL Query Language Tutorial&quot;</td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>“SPARQL Protocol Tutorial&quot;</td>
</tr>
</tbody>
</table>
Semantics of UNION

- UNION is a **binary operator**.

- Alternative graph patterns that are combined by UNION are processed **independently** of each other and the results are combined using (set-theoretic) union.
Semantics of UNION (cont’d)

• The query

```sql
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
  UNION
}
```

gives a result that is the same as the set-theoretic union of the results of the following two queries:

```sql
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
SELECT ?title

PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
```
Semantics of UNION (cont’d)

• We have to be careful whether or not to use the **same variables** in each alternative (as we did in the previous query). This decision depends on what we want to compute.
Example

• Consider now the following query where different variables are used for title:

```sparql
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?x ?y
```

• Result:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SPARQL (Updated)&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;SPARQL Protocol Tutorial&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;SPARQL&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;SPARQL Query Language Tutorial&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Properties of UNION

- **Precedence and associativity:**
  - UNION is left-associative.
  - UNION and OPTIONAL have the same precedence.
  - UNION has higher precedence than conjunction (i.e., juxtaposition of patterns).

- **Commutativity:**
  - \( P \ UNION \ Q \) is equivalent to \( Q \ UNION \ P \)

- **Associativity property:**
  - \( \{P \ UNION \ Q\} \ UNION \ R \) is equivalent to \( P \ UNION \ \{Q \ UNION \ R\} \)
Examples of Combining UNION and OPTIONAL

{ {s1 p1 o1} UNION {s2 p2 o1}
  OPTIONAL {s3 p3 o3}
}

is equivalent to

{ { {s1 p1 o1} UNION {s2 p2 o1}
  } OPTIONAL {s3 p3 o3}
}

Examples (cont’d)

\[
\{ \{ s_1 \ p_1 \ o_1 \} \ \text{OPTIONAL} \ \{ s_2 \ p_2 \ o_1 \} \\
\text{UNION} \ \{ s_3 \ p_3 \ o_3 \} \ \text{OPTIONAL} \\
\{ s_4 \ p_4 \ o_4 \} \ \text{OPTIONAL} \ \{ s_5 \ p_5 \ o_5 \} \\
\}
\]

is equivalent to

\[
\{ \{ \{ s_1 \ p_1 \ o_1 \} \ \text{OPTIONAL} \ \{ s_2 \ p_2 \ o_1 \} \\
\text{UNION} \ \{ s_3 \ p_3 \ o_3 \} \\
\} \ \text{OPTIONAL} \ \{ s_4 \ p_4 \ o_4 \} \\
\} \ \text{OPTIONAL} \ \{ s_5 \ p_5 \ o_5 \} \\
\}
\]
Examples of Combining UNION and conjunction

\{ \{s1 \ p1 \ o1\} \ UNION \ {s2 \ p2 \ o1}\ \\
    \{s3 \ p3 \ o3\}\ \\
\} 

is equivalent to

\{ \{\ {s1 \ p1 \ o1\} \ UNION \ {s2 \ p2 \ o1}\ \} \\
    \{s3 \ p3 \ o3\\}
\}

- See the difference in the results of Queries 1 and 2 below.
Example

• Data:

@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .


Example (cont’d)

• **Query 1:**

```sparql
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
WHERE {
  UNION
}
```

• **Answer:**

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/book1">http://example.org/book1</a></td>
<td>&quot;Semantic Web&quot;</td>
<td></td>
</tr>
<tr>
<td><a href="http://example.org/book3">http://example.org/book3</a></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td><a href="http://example.org/book2">http://example.org/book2</a></td>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>
Example (cont’d)

• Query 2:

PREFIX ex:  <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
WHERE {
}

• Answer:

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/book3">http://example.org/book3</a></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td><a href="http://example.org/book2">http://example.org/book2</a></td>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>
Semantics of SPARQL

- The formal semantics of SPARQL can be found in the W3C specification (http://www.w3.org/TR/rdf-sparql-query/#sparqlDefinition).

- We prefer to discuss the semantics and expressive power of SPARQL following the papers
  Available from http://www.dcc.uchile.cl/~cgutierr/papers/

- We will use the presentation from the tutorial
Evaluation of SPARQL queries

• We can evaluate SPARQL queries by translating them into the algebraic language of Perez et al. that we have just presented.
Example Query

PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>

  FILTER (?price < 30)
  { ?book dc:creator ex:Smith . } UNION
  { ?book dc:creator ex:Jones . }
}
Translation into Algebra

( ( ( {{?book, ns:price, ?price}}
    FILTER (?price < 30)
 )
   OPT
   {{?book, dc:title, ?title}}
 )
 AND
 ( {{?book, dc:creator, ex:Smith}}
   UNION
   {{?book, dc:creator, ex:Jones}}
 )
)
General Method of Translation

- **Identify the BGPs.** These are the atomic operands (the leafs of the corresponding parse tree of the algebra expression).

- Proceeding from the innermost to the outermost patterns, **use the precedence and associativity of operators** to obtain the algebra expression.
Readings

• Chapter 7 of the book “Foundations of Semantic Web Technologies”.

• The W3C Candidate Recommendation “SPARQL Query Language for RDF” from http://www.w3.org/TR/rdf-sparql-query/.

• The SPARQL tutorial given at ESWC 2007 available from http://axel.deri.ie/%7Eaxepol/sparqltutorial/ especially Unit 2 (SPARQL formalization).

• The two papers on the semantics of SPARQL cited earlier.