Tutorial 2

Despina-Athanasia Pantazi
18-11-2024

- SPARQL 1.1
- Aggregates
- Subqueries

SPARQL 1.1

- The newest version of SPARQL is SPARQL 1.1 with support for:
  - **New query features**:
    - Aggregate functions
    - Subqueries
    - Negation
    - Expressions in the SELECT clause
    - Property Paths
    - Assignment
    - A short form for CONSTRUCT
    - An expanded set of functions and operators
  - **Updates**
  - **Federated queries**
  - …

- See the web page of the SPARQL Working Group for more information: http://www.w3.org/2009/sparql/wiki/Main_Page

Aggregates

- **Aggregate functions** can be used to do computations over groups of solutions that satisfy certain graph patterns. By default a solution set consists of a single group, containing all solutions.
- Grouping is specified using the `GROUP BY` clause. It specifies the key variables to use to partition the solutions into groups.
- The `HAVING` clause can also be used to constrain grouped solutions in the same way `FILTER` constrains ungrouped ones.
- The following aggregate functions are allowed: `COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT`, and `SAMPLE`.

Aggregates

- `COUNT:` Counts the number of times the specified value is bound to the given variable.
- `SUM:` Adds the specified values.
- `MIN, MAX:` Returns the minimum/maximum value from the specified set of values.
- `AVG:` Calculates the average value for a numeric expression.
- `GROUP_CONCAT:` Performs a string concatenation of all of the values that are bound to the given variable.
- `SAMPLE:` Returns an arbitrary value from the specified set of values.

Example: Aggregates

- **Data:**

```
@prefix : <http://books.example/> .

:org1 :affiliates :auth1, :auth2 .
:auth1 :writesBook :book1, :book2 .
:book1 :price 9 .
:book2 :price 5 .
:auth2 :writesBook :book3 .
:book3 :price 7 .

:org2 :affiliates :auth3 .
:auth3 :writesBook :book4 .
:book4 :price 7 .
```

Example (cont'd)

- **Query 1:** Find how **many authors** are affiliated with each organization. Output the organization id and the amount of the authors per organization.

```
prefix : <http://books.example/>

SELECT ?org (COUNT(?auth) as ?count)
WHERE {
    ?org :affiliates ?auth .
}
GROUP BY ?org
```

Example (cont'd)

- **Query 1:** Find how **many authors** are affiliated with each organization. Output the organization id and the amount of the authors per organization.

```
prefix : <http://books.example/>


SELECT ?org (COUNT(?auth) as ?count)
WHERE {
    ?org :affiliates ?auth .
}
GROUP BY ?org
```

- **Result:**

org	count
<http://books.example/org1>	2
<http://books.example/org2>	1

Example (cont'd)

- **Query 2:** Find the most expensive book of each author. Output the author id and the price of their most expensive book.

```
prefix book: <http://books.example/>

SELECT ?auth (MAX(?price) AS ?maxprice)
WHERE {
      ?auth book:writesBook ?book .
     ?book book:price ?price
}
GROUP BY ?auth
```

Example (cont'd)

- **Query 2:** Find the most expensive book of each author. Output the author id and the price of their most expensive book.

```
prefix book: <http://books.example/>

SELECT ?auth (MAX(?price) AS ?maxprice)
WHERE {
      ?auth book:writesBook ?book .
     ?book book:price ?price
}
GROUP BY ?auth
```

Result:

auth	maxprice
<http://books.example/auth1>	9
<http://books.example/auth2>	7
<http://books.example/auth3>	7

Example (cont'd)

- **Query 3:** Find the **total price** of books written by authors affiliated with some organization. Output the organization id and total price only if the total price is greater than 10.

```
SELECT ?org (SUM(?lprice) AS ?totalPrice)
WHERE { ?org :affiliates ?auth .
        ?auth :writesBook ?book .
        ?book :price ?lprice . }
GROUP BY ?org
HAVING (SUM(?lprice) > 10)
```

Example (cont'd)

- **Query 3:** Find the **total price** of books written by authors affiliated with some organization. Output organization id and total price only if the total price is greater than 10.

```
SELECT ?org (SUM(?lprice) AS ?totalPrice)
WHERE { ?org :affiliates ?auth .
        ?auth :writesBook ?book .
        ?book :price ?lprice . }
GROUP BY ?org
HAVING (SUM(?lprice) > 10)
```

- **Result:**

org	totalPrice
<http://books.example/org1>	21

Subqueries

- Subqueries are a way to **embed SPARQL queries inside other queries** to allow the expression of requests that are not possible otherwise.
- Subqueries are useful when combining limits and aggregates with other constructs.
- Subqueries are evaluated first and then the outer query is applied to their results.
- Only variables projected out of the subquery (i.e., appearing in its SELECT clause) will be visible to the outer query.

Example: Subqueries

Data:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix : <http://sales.com/> .

:sale1 a :Sale ; :company :c1 ; :amount 7500^^xsd:integer  ; :year "2011" .
:sale2 a :Sale ; :company :c1 ; :amount 17000^^xsd:integer ; :year "2011" .
:sale3 a :Sale ; :company :c1 ; :amount 5500^^xsd:integer ; :year "2012" .
:sale4 a :Sale ; :company :c1 ; :amount 7000^^xsd:integer ; :year "2012" .
:sale5 a :Sale ; :company :c2 ; :amount 3000^^xsd:integer ; :year "2011" .
:sale6 a :Sale ; :company :c2 ; :amount 4000^^xsd:integer ; :year "2011" .
:sale7 a :Sale ; :company :c2 ; :amount 5000^^xsd:integer ; :year "2012" .
:sale8 a :Sale ; :company :c2 ; :amount 6000^^xsd:integer ; :year "2012" .
```

Example Subqueries

- **Query:** Find companies that increased their sales from 2011 to 2012 and the amount of increase.

```
PREFIX : <http://sales.com/>
SELECT ?c ((?total2012 - ?total2011) AS ?increase)
WHERE {
    { SELECT ?c (SUM(?m) AS ?total2012)
      WHERE { ?s a :Sale ; :company ?c ;
                   :amount ?m ; :year: "2012" . }
      GROUP BY ?c
    } .
    { SELECT ?c (SUM(?m) AS ?total2011)
      WHERE { ?s a :Sale ; :company ?c ;
                   :amount ?m ; :year: "2011" . }
      GROUP BY ?c
    } .
FILTER (?total2012 > ?total2011)
}
```

Example Subqueries

- **Results:**

c	increase
`<http://sales.com/c2>`	`"4000"^^<http://www.w3.org/2001/XMLSchema#integer>`