ISIT315
Week 7

SPARQL 2

Recap: Structure of a SPARQL Query

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
ORDER BY ...
```

SPARQL built-in filter functions

- Logical: !, &&, ||
- Math: +, -, *, /
- Comparison: =, !=, >, <, ...
- SPARQL tests: isURI, isBlank, isLiteral, bound
- SPARQL accessors: str, lang, datatype
- Other: sameTerm, langMatches, regex

Example 1

```
SELECT ?actor
WHERE {?actor :playedIn :EastOfEden .
   FILTER (?birthday>”1930-01-01”^^xsd:date)}
```

What is the output?

Example 2

```
SELECT ?actor
WHERE {?actor :playedIn :EastOfEden .
            ?actor :bornOn ?birthday .
  FILTER (?birthday>”1930-01-01”^^xsd:date)}
```

Example 3

```
SELECT ?actor
WHERE {?actor :playedIn :EastOfEden .
       ?actor :bornOn ?birthday .
  FILTER (?birthday>”Jan 1, 1960”^^xsd:date)
  FILTER (?birthday<”Dec 31, 1969”^^xsd:date)}
```

Notes

- Cannot reference a variable in the FILTER if that variable has not been referenced in the graph pattern
- If multiple filters are used, all tests must be TRUE to return a result

Binding

- A value is <u>bound</u> to a variable
  - Example ?actor is bound to JamesDean
- A result is returned only when a value is bound to a variable, else it will not

ASK

- Asking a question
  - Return TRUE or FALSE

```
ASK WHERE {
   ?any :playedIn :GIANT .
   ?any :bornOn ?birthday .
   FILTER (?birthday > “1950-01-
     01”^^xsd:date)}
```

Example 1

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

ASK  { ?x foaf:name  "Alice" }
```

Output

- TRUE

Example 2

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

```
ASK  { ?x foaf:name  "Alice" ;
          foaf:mbox  <mailto:alice@work.example> }
```

Output

- FALSE

Limits and Ordering

```
SELECT ?title ?date
WHERE {
        :JamesDean :playedIn ?movie .
        ?movie rdfs:label ?title .
         ?movie dc:date ?date .}
ORDER BY ?title
```

Another example

```
SELECT ?title ?date
WHERE {
      :JamesDean :playedIn ?movie .
      ?movie rdfs:label ?title .
       ?movie dc:date ?date .}
ORDER BY ?title
LIMIT 1
```

Another example

```
SELECT ?title
WHERE {
        :JamesDean :playedIn ?movie .
        ?movie rdfs:label ?title .
         ?movie dc:date ?date .}
ORDER BY DESC ?title
```

Aggregates and grouping

- Aggregate functions
  - COUNT
  - MIN
  - MAX
  - AVG
  - SUM

Example 1

```
SELECT (COUNT (?movie) AS ?howmany)
WHERE {:JamesDean ?playedIn ?movie .}
```

Consider the following example

Company	Amount	Year
ACME	$1250	2010
PRIME	$3000	2009
ABC	$2500	2009
ABC	$2800	2010
PRIME	$1950	2010
ACME	$2500	2009
ACME	$3100	2010
ABC	$1500	2009
ACME	$1250	2009
PRIME	$2350	2009
PRIME	$1850	2010

In Triples

Each row has 4 triples

```
:row1 a :Sale .
:row1 :company :ACME .
:row1 :amount : 1250 .
:row1 :year 2010
```

SUM

```
SELECT (SUM (?val) AS ?total)
WHERE {
          ?s a: Sale .
          ?s :amount ?val }
```

What is the output?

GROUP BY

```
SELECT ?year (SUM (?val) AS ?total)
WHERE {
          ?s a: Sale .
          ?s :amount ?val .
          ?s :year ?year }
GROUP BY ?year
```

What is the output?

Another example

```
SELECT ?year ?company (SUM (?val) AS ?total)
WHERE {
          ?s a: Sale .
          ?s :amount ?val .
          ?s :year ?year .
           ?s :company ?company}
GROUP BY ?year ?company
```

What is the output?

HAVING

```
SELECT ?year ?company (SUM (?val) AS ?total)
WHERE {
            ?s a: Sale .
            ?s :amount ?val .
            ?s :year ?year .
             ?s :company ?company}
GROUP BY ?year ?company
HAVING (?total > 5000)
```

What is the output?

UNION

- Combines two graph patterns

```
SELECT ?actor
WHERE {
       {?actor :playedIn :Giant .}
      UNION
       {?actor :playedIn :RebelWithoutaCause .}
      }
```

Exercise 1

```
:John a :Man .
:Joe a :Man .
:Eunice a :Woman .
:Maria a :Woman .
:Caroline a: Woman .
:Ted a :Man .
:Caroline :hasFather :John .
:John :hasFather :Joe .
:Maria :hasMother :Eunice .
:Maria :hasFather :Sargen .
:Ted :hasSister :Eunice .
```

Write query for

- To relate father to son
- To define “uncle”

Exercise 2

```
:John a :Man .
:Joe a :Man .
:Eunice a :Woman .
:Maria a :Woman .
:Caroline a: Woman .
:Ted a :Man .
:Caroline :hasFather :John .
:John :hasFather :Joe .
:Maria :hasMother :Eunice .
:Maria :hasFather :Sargen .
:Ted :hasSister :Eunice .
:Joe :hasSon :Robert
:Joe :hasSon :Ted
:Ted :hasSon :Patrick
```

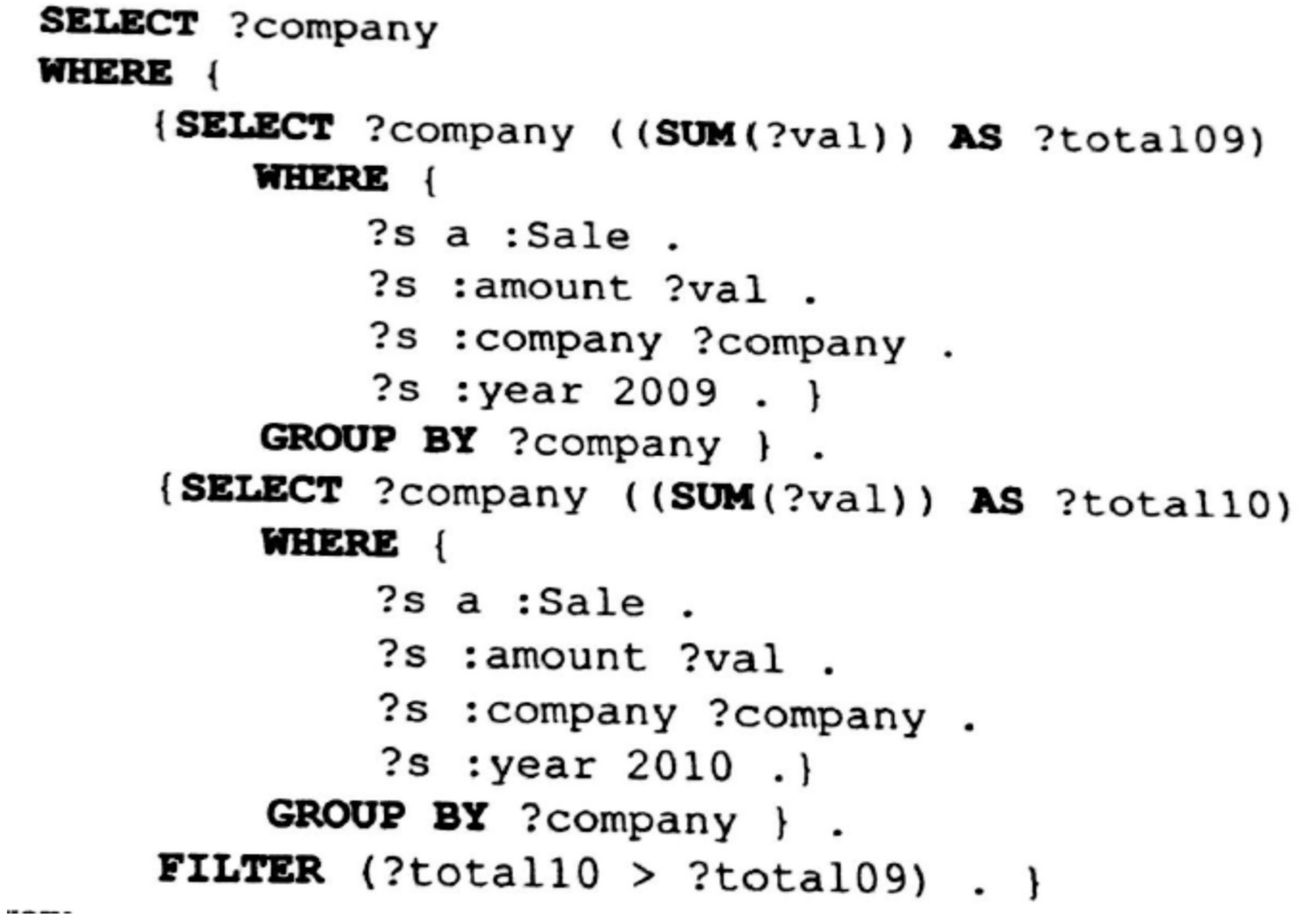
Write query to find

- hasParent
- All members of Joe’s family
- Grandchildren of Joe

SUBQUERIES

- Query within a query
- Generally subquery is not required in SPARQL because SPARQL graph pattern can include arbitrary connections between variables and resource identifiers
- However subqueries are useful when combining limits and aggregates with other graph patterns.

Example: Subquery to compute total sales for 2009 and 2010

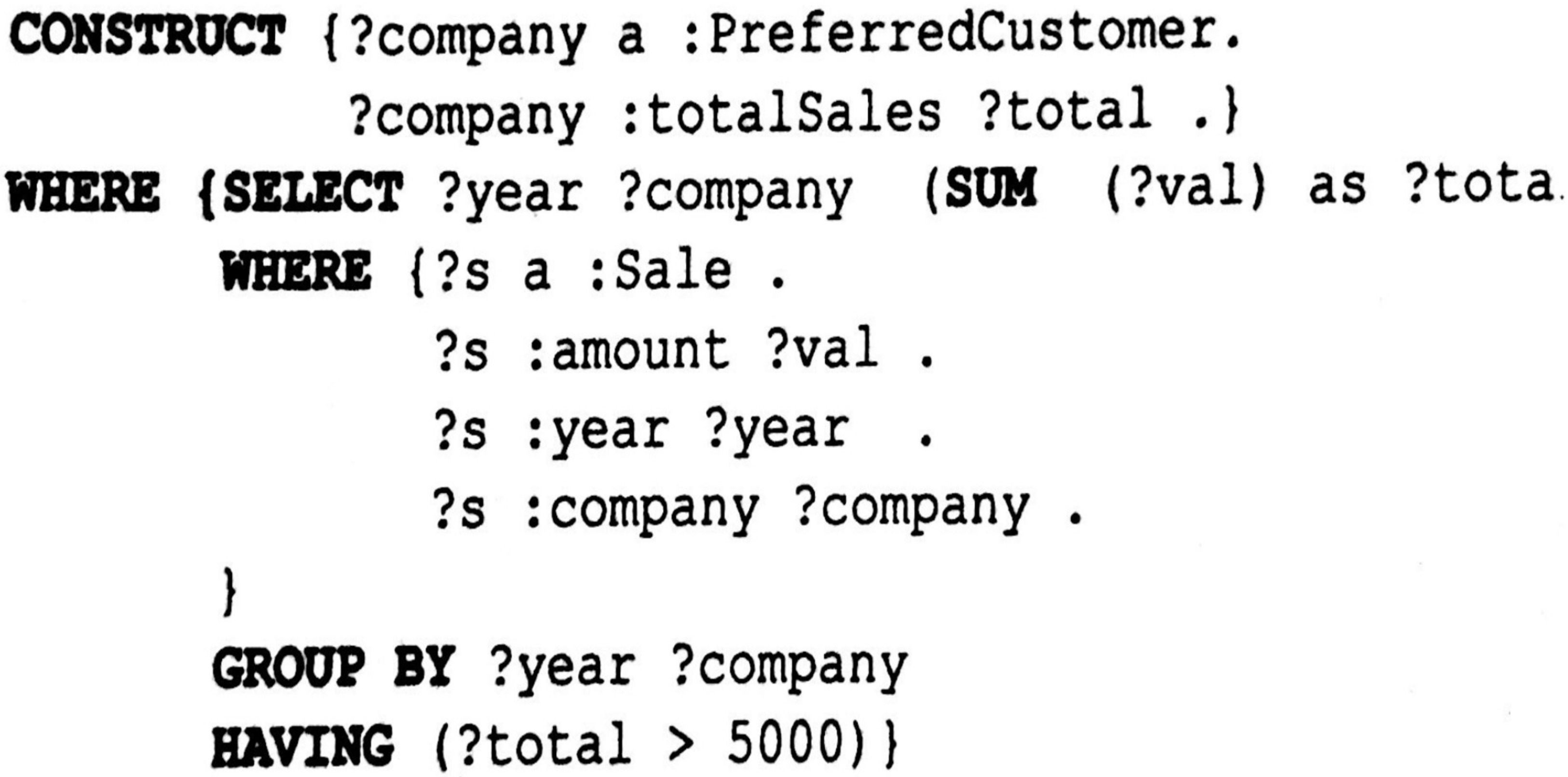


```
SELECT ?company
WHERE {
    {SELECT ?company ((SUM(?val)) AS ?total09)
        WHERE {
            ?s a :Sale .
            ?s :amount ?val .
            ?s :company ?company .
            ?s :year 2009 . }
        GROUP BY ?company } .
    {SELECT ?company ((SUM(?val)) AS ?total10)
        WHERE {
            ?s a :Sale .
            ?s :amount ?val .
            ?s :company ?company .
            ?s :year 2010 .}
        GROUP BY ?company } .
    FILTER (?total10 > ?total09) . }
```

Output

```
?company
ACME
```

Another example: Using subquery in CONSTRUCT



```
CONSTRUCT {?company a :PreferredCustomer.
           ?company :totalSales ?total .}
WHERE {SELECT ?year ?company  (SUM  (?val) as ?tota
       WHERE {?s a :Sale .
              ?s :amount ?val .
              ?s :year ?year  .
              ?s :company ?company .
       }
       GROUP BY ?year ?company
       HAVING (?total > 5000)}
```

Output

```
:PRIME a :PreferredCustomer .
:PRIME :totalSales 5350.00 .
```

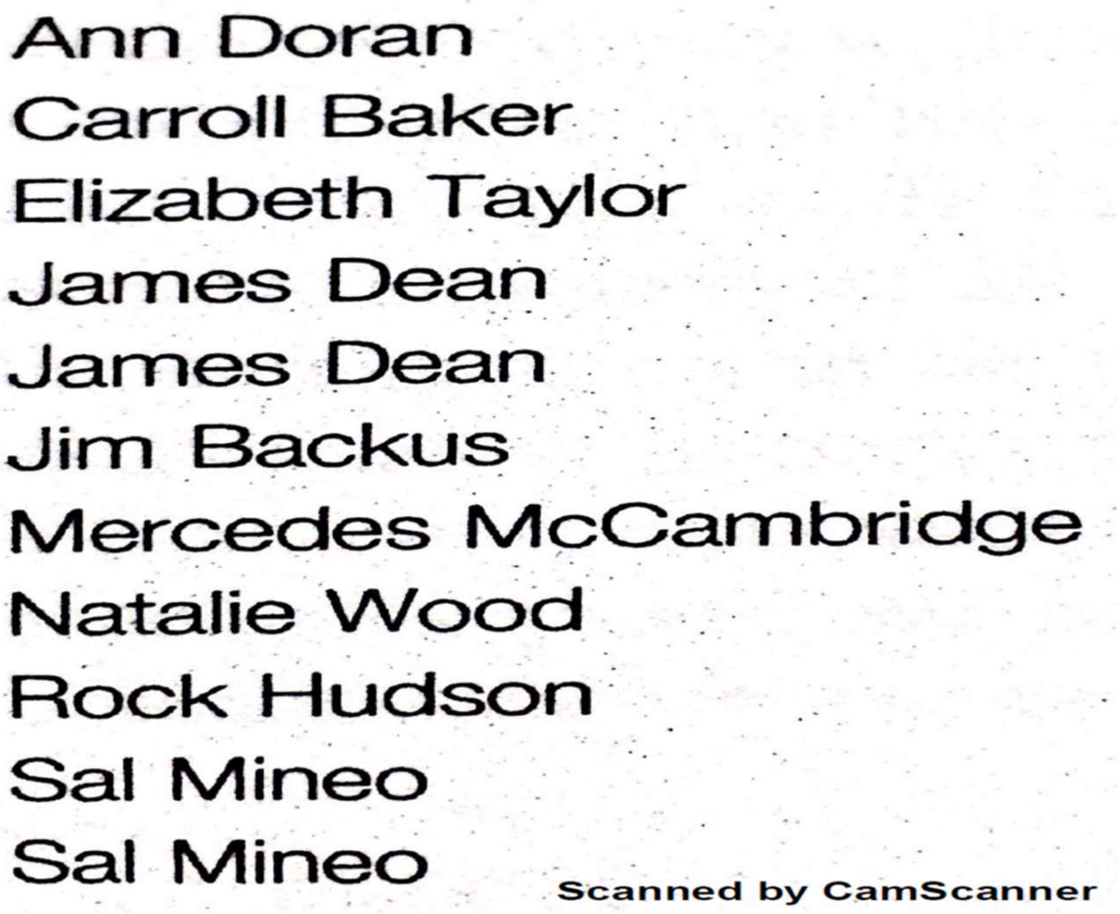
UNION

- Combines two graph patterns
- Variables in each pattern takes values independently but the results are combined together

Example

```
SELECT ?actor
WHERE {
        {?actor :playedIn : Giant .}
        UNION
         {?actor :playedIn : RebelWithoutaCause .}
        }
```

Output



```
Ann Doran
Carroll Baker
Elizabeth Taylor
James Dean
James Dean
Jim Backus
Mercedes McCambridge
Natalie Wood
Rock Hudson
Sal Mineo
Sal Mineo
```

Scanned by CamScanner

Question

- How do you remove duplicate names?

Exercise 1

```
:John a :Man .
:Joe a :Man .
:Eunice a :Woman .
:Maria a :Woman .
:Caroline a: Woman .
:Ted a :Man .
:Caroline :hasFather :John .
:John :hasFather :Joe .
:Maria :hasMother :Eunice .
:Maria :hasFather :Sargen .
:Ted :hasSister :Eunice .
```

Write query for:

- To relate father to son
- To define Uncle in terms of siblings and parents

SPARQL Endpoint

- A server for the SPARQL protocol
  - accepts queries and returns results via HTTP.
    - Generic endpoints will query any Web-accessible RDF data
    - Specific endpoints are hardwired to query against particular datasets
- Endpoint is identified with a URL and provides flexible access to its data set

Various output formats

- The results of SPARQL queries can be returned and/or rendered in a variety of formats:
  - XML. SPARQL specifies an XML vocabulary for returning tables of results.
  - JSON. A JSON "port" of the XML vocabulary, particularly useful for Web applications.
  - CSV/TSV. Simple textual representations ideal for importing into spreadsheets
  - RDF. Certain SPARQL result clauses trigger RDF responses, which in turn can be serialized in a number of ways (RDF/XML, N-Triples, Turtle, etc.)
  - HTML. When using an interactive form to work with SPARQL queries. Often implemented by applying an XSL transform to XML results.

The intention of SPARQL endpoints

- Give other people and organisations access to your data in a very flexible way
- Eventually realise the potential of federated SPARQL whereby several SPARQL Endpoints are combined to allow complex queries to be run across a number of datasets
- They are open for use by a large and varied audience

Challenges of using SPARQL endpoint

- Intermittently available or not available

Examples of SPARQL Query Editors

- Rasqal
- Virtuoso
- Flint SPARQL query editor