Semantic Web Engineering

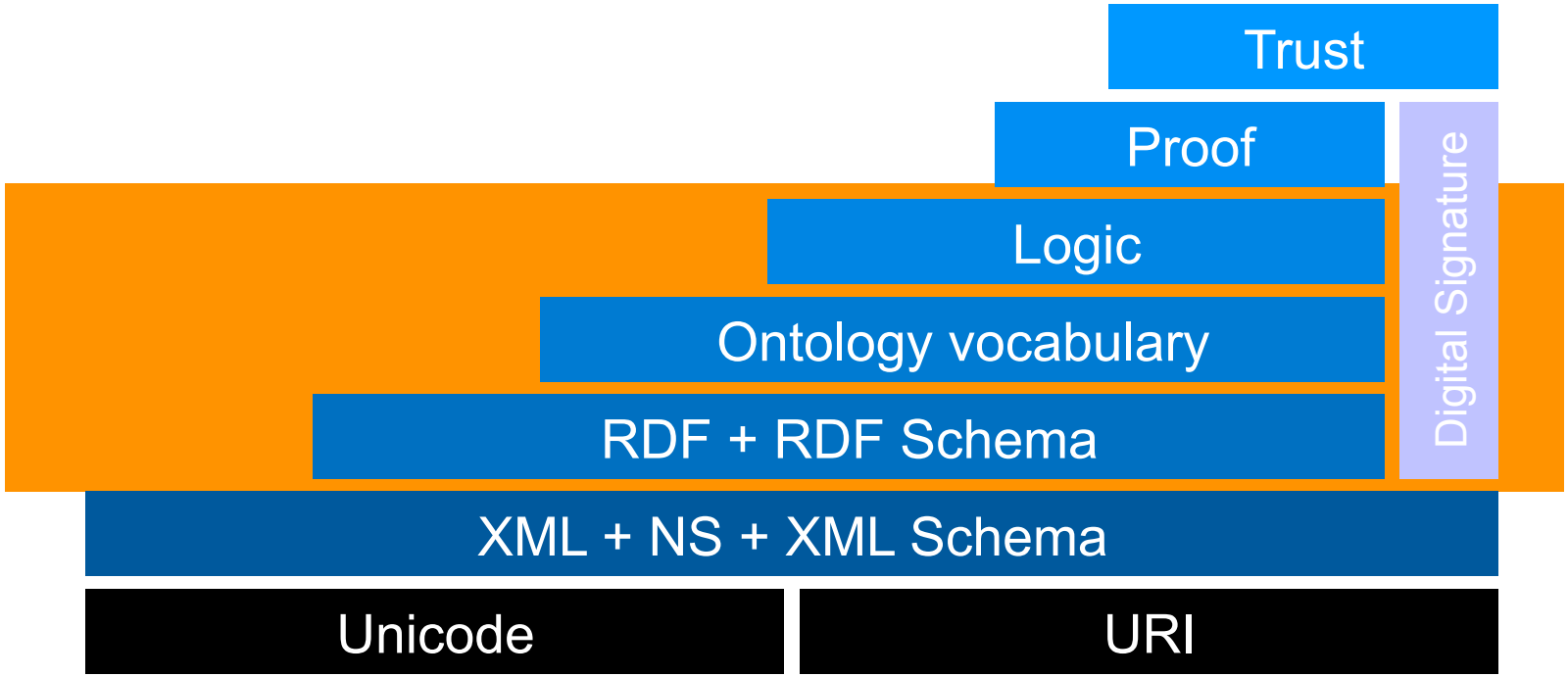
Matthias Hert
hert@ifi.uzh.ch

Fr. 10:15-12:00, Room 2.A.10

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Web Ontology Language



RDF Query Language

- **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
  - SPARQL Query Language for RDF
    http://www.w3.org/TR/rdf-sparql-query/
    Recommendation 15 January 2008
  - SPARQL Protocol for RDF
    http://www.w3.org/TR/rdf-sparql-protocol/
    Recommendation 15 January 2008
- SPARQL is influenced by:
  - SQL-like: RDQL/Squish, SeRQL, RDFDB QL, RQL, ...
  - XPath-like: Versa, RDFPath
  - Rules-like: N3QL, Triple, DQL, OWL-QL, ...
  - Language-like: Algae2, Fabl, Abeline
  - Using XML: XSLT, XPath, XQuery

Querying XML compared to Querying RDF

- Why does RDF need ist own query language?

Concept	XML	RDF
Model	Document or Tree or Infoset	Set of Triples = RDF Graph
Atomic Units	Elements, Attributes, Text	Triples, URIs, Blank Nodes, Text
Identifiers	Element/Attribute names QNames, IDs XPointers / XPaths	URIs
Described by	DTDs, XML Schema Relax NG	RDF Schema, OWL

SPARQL is Query Language and Protocol

- SPARQL - Query Language
  - An RDF data access query language
  - Data access means reading information, not writing (updates)
  - Outline query model is graph patterns
- SPARQL - Protocol
  - Services running SPARQL queries over a set of graphs
  - A transport protocol for invoking the service
  - Describing the service with Web Service technologies

Turtle RDF syntax - URIs and Blank Nodes

- Turtle - Terse RDF Triple Language
  http://www.dajobe.org/2004/01/turtle/
- URIs
  - Enclosed in <>
  - or
  - `@prefix prefix: <http://....>`
  - `prefix:name`
  - in the style of XML QNames as a shorthand for the full URI
- Blank Nodes
  - `_:name`
  - or
  - `[]` for a Blank Node used once

Turtle RDF Syntax - Triples and Abbreviations

- Triples separated by .
  :a :b :c . :d :e :f .
- Common triple subject and predicate:
  :a :b :c, :d .
  which is the same as :a :b :c . :a :b :d .
- Common triple subject:
  :a :b :c; :d :e . which is the same as: :a :b :c . :a :d :e .
- Blank node as a subject
  :a :b [:c :d] which is the same as: :a :b _:x . _:x :c :d .
  for blank node _:x
- RDF Collections
  :a :b (:c :d :e :f)
  which is short for many triples

SPARQL Query

- A SPARQL query is matched against the RDF graph.
- SPARQL has four query result forms. These result forms use the solutions from pattern matching to form result sets or RDF graphs. The query forms are:
  - SELECT
    - Returns all, or a subset of, the variables bound in a query pattern match in tabular format (similar to SQL).
  - CONSTRUCT
    - Returns an RDF graph constructed by substituting variables in a set of triple templates.
  - DESCRIBE
    - Returns an RDF graph that describes the resources found.
  - ASK
    - Returns a boolean indicating whether a query pattern matches or not.

SPARQL Demo

- SPARQL Demo using ARQ.
  - ARQ is a query engine for Jena that supports the SPARQL RDF Query language.
  - http://jena.sourceforge.net/ARQ/
  - Demo inspired by:
    http://www.w3.org/2004/Talks/17Dec-sparql/
- Online Demo
  - http://sparql.org/query.html
- Download ARQ
  - http://jena.sourceforge.net/ARQ/download.html
- Twinkle: A SPARQL Query Tool
  - http://www.ldodds.com/projects/twinkle/

SPARQL Simple Query

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample1.ttl)

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name  "Alice" .
_:a  foaf:mbox  <mailto:alice@example.net> .
_:b  foaf:name  "Bob" .
```

Query (q1.rq)

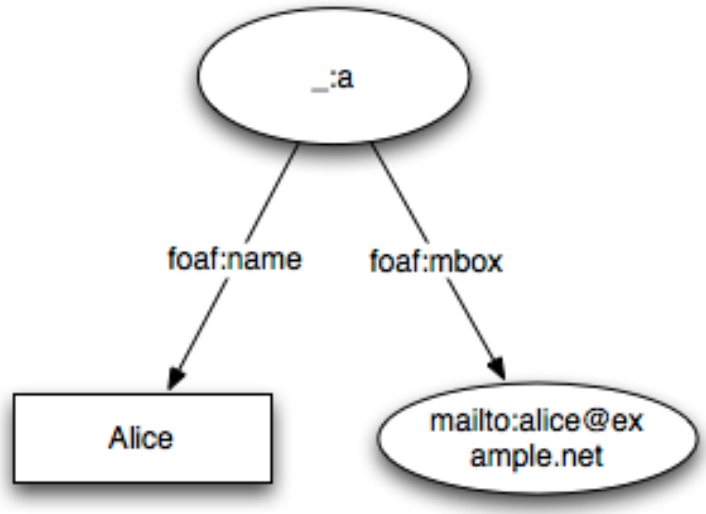
```
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x  foaf:name  ?name }
```

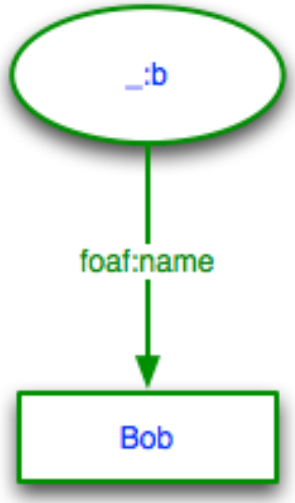
Result

```
-----------
| name     |
===========
| "Bob"    |
| "Alice"  |
-----------
```

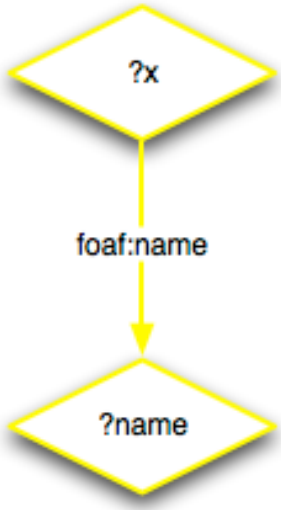
SPARQL Simple Query

Data





Query



Result

?x → _:a, ?name → “Alice”
?x → _:b, ?name → “Bob”

Matching

- *Matches the graph* means find a set of bindings such that the substitution of variables for values creates a triple that is in the set of triples making up the graph.
- Solution 1: variable *x* has value blank node _:b and variable *name* has value "Bob"
  Triple `_:b foaf:name "Bob"` is in the graph.
- Solution 2: variable *x* has value blank node _:a and variable *name* has value *"Alice"*
  Triple `_:a foaf:name "Alice"` is in the graph.
- No order of solutions in this query.

PREFIX and BASE

- The PREFIX keyword binds a prefix to a namespace.
- The BASE keyword defines the Base URI used to resolve relative URIs.
- The following examples express the same queries:

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

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

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

Query Results 1/2

- Query results are returned in SPARQL Variable Binding Results XML Format.
  - W3C Recommendation 15 January 2008
  - http://www.w3.org/TR/rdf-sparql-XMLres/
- Call to the ARQ SPARQL implementation to see the result in XML format:
  - `sparql --data=sample1.ttl --query=q1.rq --results=xml`
  - Default result format is `text`.

Query Results 2/2

- Result for the simple query before:

```
<?xml version="1.0"?>
<sparql xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:xs="http://www.w3.org/2001/XMLSchema#"
        xmlns="http://www.w3.org/2005/sparql-results#" >
  <head>
    <variable name="name"/>
  </head>
  <results ordered="false" distinct="false">
    <result>
      <binding name="name">
        <literal>Bob</literal>
      </binding>
    </result>
    <result>
      <binding name="name">
        <literal>Alice</literal>
      </binding>
    </result>
  </results>
</sparql>
```

SPARQL Simple Query

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample1.ttl)

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name  "Alice" .
_:a  foaf:mbox  <mailto:alice@example.net> .
_:b  foaf:name  "Bob" .
```

Query (q1.rq)

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

Result

```
-----------
| name    |
===========
| "Bob"   |
| "Alice" |
-----------
```

SPARQL Simple Query

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample1.ttl)

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name  "Alice" .
_:a  foaf:mbox  <mailto:alice@example.net> .
_:b  foaf:name  "Bob" .
```

Query (q2.rq)

```
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?x
WHERE { ?x  foaf:name  "Alice" }
```

Result

SELECT * is an abbreviation that selects all of the named variables.

```
-----------
| name    |
===========
| _:a     |
-----------
```

SPARQL Simple Query

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample1.ttl)

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name  "Alice" .
_:a  foaf:mbox  <mailto:alice@example.net> .
_:b  foaf:name  "Bob" .
```

Query (q3.rq)

```
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
  ?person  foaf:name  ?name .
  ?person  foaf:mbox  <mailto:alice@example.net> .
}
```

Result

```
-----------
| name    |
===========
| "Alice" |
-----------
```

Syntax of Literal Terms

- The query terms can be literals which are a string (enclosed in quotes, either double quotes `""` or single quotes `''`), with either an optional language tag (introduced by `@`) or an optional datatype URI or qname (introduced by `^^`). As a convenience, integers can be written directly and are interpreted as typed literals of datatype `xsd:integer`; decimal numbers are interpreted as `xsd:decimal` and a numer with an exponent is interpreted as an `xsd:double`. Values of type `xsd:boolean` can also be written as `true` or `false`.
- Examples of literal syntax in SPARQL include:
  - `"chat"`
  - `"chat"@fr` with language tag `"fr"`
  - `"xyz"^^<http://example.org/ns/userDatatype>`
  - `"abc"^^appNS:appDataType`
  - `1`, which is the same as `"1"^^xsd:integer`
  - `1.3`, which is the same as `"1.3"^^xsd:decimal`
  - `1.0e6`, which is the same as `"1.0e6"^^xsd:double`
  - `true`, which is the same as `"true"^^xsd:boolean`
  - `false`, which is the same as `"false"^^xsd:boolean`

Constraints using `Filter` 1/2

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample2.ttl)

```
@prefix  dc:     <http://purl.org/dc/elements/1.1/> .
@prefix  stock:  <http://example.org/stock#> .
@prefix  inv:    <http://example.org/inventory#> .

stock:book1  dc:title      "SPARQL Query Language Tutorial" .
stock:book1  inv:price     10 .
stock:book1  inv:quantity  3 .

stock:book2  dc:title   "SPARQL Query Language (2nd ed)" .
stock:book2  inv:price  20 ;  inv:quantity  5 .

stock:book3  dc:title   "Moving from SQL to SPARQL" .
stock:book3  inv:price  5 ;  inv:quantity  0 .

stock:book4  dc:title   "Applying XQuery" .
stock:book4  inv:price  20 ;  inv:quantity  8 .
```

Constraints using `Filter` 2/2

Available functions and operations
http://www.w3.org/TR/rdf-sparql-query/#OperatorMapping

Query (q4.rq)

```
PREFIX  dc:     <http://purl.org/dc/elements/1.1/>
PREFIX  stock:  <http://example.org/stock#>
PREFIX  inv:    <http://example.org/inventory#>

SELECT ?book ?title
WHERE {
  ?book  dc:title     ?title .
  ?book  inv:price    ?price .  FILTER (?price < 15) .
  ?book  inv:quantity ?num .    FILTER (?num > 0) .
}
```

Result

```
-------------------------------------------------
| book         | title                          |
=================================================
| stock:book1  | "SPARQL Query Language Tutorial" |
-------------------------------------------------
```

Construct Queries 1/2

- CONSTRUCT results are made from variable substitutions into the pattern and return RDF graphs.

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample3.ttl)

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

_:a  foaf:givenname  "Alice" .
_:a  foaf:knows      _:b .
_:a  foaf:mbox       <mailto:alice@example.net> .

_:b  foaf:givenname  "Bob" ;
     foaf:mbox       <mailto:bob@example.com> ;
     foaf:knows      _:c .

_:c  foaf:givenname  "Chuck" ;
     foaf:mbox       <mailto:chuck@example.com> ;
     foaf:knows      _:a .
```

Construct Queries 2/2

Query "Who does Alice know?" (q5.rq)

```
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX  pim:   <http://www.w3.org/2000/10/swap/pim/contact#>
CONSTRUCT {
  ?friend  pim:fullName      ?name .
  ?friend  pim:emailAddress  ?mbox . }
WHERE {
  ?alice   foaf:knows      ?friend .
  ?alice   foaf:givenname  ?a_name  FILTER (?a_name = "Alice") .
  ?friend  foaf:givenname  ?name.
  ?friend  foaf:mbox       ?mbox .
}
```

Result

```
<rdf:Description>
    <pim:fullName>Bob</pim:fullName>
    <pim:emailAddress rdf:resource="mailto:bob@example.com"/>
</rdf:Description>
```

Optional Pattern Matching 1/4

- Optional parts of the graph pattern may be specified syntactically with the OPTIONAL keyword applied to a graph pattern.

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample4.ttl)

```
@prefix  foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix  vCard:  <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:a  vCard:firstName  "Alice" ;
     vCard:lastName   "Smith" ;
     foaf:name        "Alice Smith" ;
     vCard:email      <mailto:alice@example.net> .

_:b  vCard:firstName  "Bob" ;
     vCard:lastName   "Bush" ;
     vCard:email      <mailto:bob@example.net> .

_:c  vCard:firstName  "Chuck" ;
     vCard:email      <mailto:chuck@example.net> .
```

Optional Pattern Matching 2/4

- `OPTIONAL` can produce solutions with unbound variables.

Query (q6.rq versus q7.rq)

```
PREFIX  foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX  vCard:  <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { ?who  foaf:mbox          ?mbox .
            ?who  foaf:givenName     ?given .
            ?who  foaf:family_name   ?family .
            ?who  foaf:name          ?full }
 WHERE { ?who  vCard:email  ?mbox .
       OPTIONAL { ?who  vCard:firstName  ?given .
                  ?who  vCard:lastName   ?family }
       OPTIONAL { ?who  foaf:name  ?full } }
```

- Statements involving unbound variables are omitted.

Result

```
[]     foaf:mbox          <mailto:chuck@example.net> .
[]     foaf:family_name   "Bush" ;
       foaf:givenName     "Bob" ;
       foaf:mbox          <mailto:bob@example.net> .
[]     foaf:family_name   "Smith" ;
       foaf:givenName     "Alice" ;
       foaf:mbox          <mailto:alice@example.net> ;
       foaf:name          "Alice Smith" .
```

Optional Pattern Matching 3/4

- Works also for `SELECT` queries.

Data (http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SemWebEng/WS05/sample5.ttl)

```
@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" .
```

Optional Pattern Matching 4/4

Query (q8.rq)

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

Result

```
-----------------------------------------
| name     | mbox                        |
=========================================
| "Bob"    |                             |
| "Alice"  | <mailto:alice@work.example> |
| "Alice"  | <mailto:alice@example.com>  |
-----------------------------------------
```

Constraints in Optional Pattern Matching

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

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

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

title	price
"SPARQL Tutorial"	
"The Semantic Web"	23

No price appears for the book with title "SPARQL Tutorial" because the optional graph pattern did not lead to a solution involving the variable price.

Joining Patterns with UNION

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

```
@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" .

_:b  dc11:title     "SPARQL Protocol Tutorial" .

_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

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

SELECT ?title
WHERE  { { ?book dc10:title  ?title } UNION { ?book dc11:title  ?title } }
```

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

`ORDER BY` 1/2

- The `ORDER BY` clause takes a solution sequence and applies ordering conditions.
- The direction of ordering is ascending by default.
- It can be explicitly set to ascending or descending by enclosing the condition in `ASC()` or `DESC()` respectively.
- If multiple conditions are given, then they are applied in turn until one gives the indication of the ordering.

ORDER BY 2/2

Examples:

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE { ?x foaf:name ?name }
ORDER BY ?name
```

```
PREFIX     :    <http://example.org/ns#>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>

SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY DESC(?emp)
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ?name DESC(?emp)
```

LIMIT

- The `LIMIT` form puts an upper bound on the number of solutions returned.
- If the number of actual solutions is greater than the limit, then at most the limit number of solutions will be returned.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE { ?x foaf:name ?name }
LIMIT 20
```

OFFSET

- OFFSET causes the solutions generated to start after the specified number of solutions. An OFFSET of zero has no effect.
- The order in which solutions are returned is initially undefined.
- Using LIMIT and OFFSET to select different subsets of the query solutions will not be useful unless the order is made predictable by using ORDER BY.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE   { ?x foaf:name ?name }
ORDER BY ?name
LIMIT   5
OFFSET  10
```

FROM

- A SPARQL query may specify the dataset to be used for matching using the `FROM` clause.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://seal.ifi.unizh.ch/fileadmin/User_Filemount/
                Vorlesungs_Folien/SemWebEng/WS05/sample1.ttl>
WHERE { ?x foaf:name ?name }
```

- Can be tested on http://www.sparql.org/sparql.html
- A SPARQL query engine is not bound to a locale dataset. The dataset can be specified with `FROM`.
  - Anyone can provide a SPARQL query engine for general use.

What's missing? — SPARQL 1.1

- SPARQL 1.0 is a basic graph pattern matching query language.
- Several features known from data query languages (e.g. SQL) are missing
- SPARQL 1.1 (Working Draft):
  - Aggregates (count, sum, …)
  - (explicit) negation
  - Subqueries
  - Expressions
  - Data updates
  - Property Paths

SPARQL 1.1: Aggregates 1/2

Data (AggregatesData.ttl)

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

:org1      :affiliates   :author1,
                         :author2 .
:org2      :affiliates   :author3 .

:author1   :writesBook   :book1,
                         :book2 .
:author2   :writesBook   :book3 .
:author3   :writesBook   :book4 .

:book1     :price        9 .
:book2     :price        5 .
:book3     :price        7 .
:book4     :price        7 .
```

SPARQL 1.1: Aggregates 2/2

Query (AggregatesQuery.rq)

```
PREFIX : <http://books.example.com/>

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

Result

```
-------------------------------------------------
| org                                | totalPrice |
=================================================
| <http://books.example.com/org1> | 21         |
-------------------------------------------------
```

SPARQL 1.1: Negation 1/2

Data (NegationData.ttl)

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

:alice  rdf:type   foaf:Person ;
        foaf:name  "Alice" .

:bob    rdf:type   foaf:Person .
```

SPARQL 1.1: Negation 2/2

Query (NegationQuery.rq)

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

SELECT ?person
WHERE
{
    ?person  rdf:type  foaf:Person .
    FILTER NOT EXISTS {
        ?person  foaf:name  ?name
    }
}
```

2 new forms of negation:
- FILTER NOT EXISTS
- MINUS

Result

```
-------------------------------------
| person                            |
=====================================
| <http://people.example.com/bob> |
-------------------------------------
```

SPARQL 1.1: Subqueries 1/2

Data (SubqueriesData.ttl)

```
@prefix : <http://people.example.com/> .

:alice   :name    "Alice",
                  "Alice Foo",
                  "A. Foo" ;
         :knows   :bob,
                  :carol .

:bob     :name    "Bob",
                  "Bob Bar",
                  "B. Bar" .

:carol   :name    "Carol",
                  "Carol Baz",
                  "C. Baz" .
```

SPARQL 1.1: Subqueries 2/2

Query (SubqueriesQuery.rq)

```
PREFIX : <http://people.example.com/>

SELECT ?friend ?anyName
WHERE {
  :alice  :knows  ?friend .
  {
    SELECT ?friend (SAMPLE(?name) AS ?anyName)
    WHERE {
      ?friend  :name  ?name .
    }
    GROUP BY ?friend
  }
}
```

Result

```
-----------------------------------------------
| friend                                | anyName  |
===============================================
| <http://people.example.com/carol> | "C. Baz" |
| <http://people.example.com/bob>   | "B. Bar" |
-----------------------------------------------
```

SPARQL 1.1: Expressions 1/2

Data (ExpressionsData.ttl)

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

:book1  dc:title     "SPARQL Tutorial" .
:book1  ns:price     42 .
:book1  ns:discount  0.1 .

:book2  dc:title     "The Semantic Web" .
:book2  ns:price     23 .
:book2  ns:discount  0 .
```

SPARQL 1.1: Expressions 2/2

Query (ExpressionsQuery.rq)

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

SELECT  ?title  (?p * (1 - ?discount) AS ?price)
WHERE {
  ?x  ns:price     ?p ;
      dc:title     ?title ;
      ns:discount  ?discount .
}
```

Result

```
-------------------------------
| title               | price |
===============================
| "The Semantic Web" | 23    |
| "SPARQL Tutorial"  | 37.8  |
-------------------------------
```

SPARQL 1.1: Property Paths 1/3

Data (PropertyPathsData.ttl)

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

:alice  foaf:name   "Alice" ;
        foaf:knows  :bob ,
                    :chuck .
:bob    foaf:name   "Bob" ;
        foaf:knows  :alice ,
                    :chuck ,
                    :dave .
:chuck  foaf:name   "Chuck" ;
        foaf:knows  :alice ,
                    :bob .
:dave   foaf:name   "Dave" ;
        foaf:knows  :bob ,
                    :ed .
:ed     foaf:name   "Ed" ;
        foaf:knows  :dave .
```

SPARQL 1.1: Property Paths 2/3

Query

```
PREFIX  :       <http://people.example.com/>
PREFIX  foaf:   <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE {
  :alice   foaf:knows   ?temp .
  ?temp    foaf:name    ?name .
}
```

Result

```
-----------
| name    |
===========
| "Chuck" |
| "Bob"   |
-----------
```

SPARQL 1.1: Property Paths 3/3

Query (PropertyPathsQuery.rq)

```
PREFIX  :      <http://people.example.com/>
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?name
WHERE {
  :alice  (foaf:knows)+/foaf:name  ?name .
  FILTER (?name != "Alice")
}
```

Result

```
-----------
| name     |
===========
| "Chuck" |
| "Bob"   |
| "Dave"  |
| "Ed"    |
-----------
```

SPARQL 1.1 Update

- *"[…] is a language to express updates to a graph store. […] is a companion language to SPARQL and is envisaged to be used in conjunction with the SPARQL 1.1 Query language."*
- Graph update:
  - `INSERT DATA / DELETE DATA`
  - `INSERT / DELETE`
  - `LOAD`
  - `CLEAR`
- Graph management:
  - `CREATE`
  - `DROP`

SPARQL 1.1: Data Updates 1/4

Data (UpdateData.ttl)

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

:alice  foaf:name   "Alice" ;
        foaf:mbox   <mailto:alice@example.net> ;
        foaf:knows  :bob .

:bob    foaf:mbox   <mailto:bob@example.net> ;
        foaf:knows  :alice .
```

SPARQL 1.1: Data Updates 2/4

Query (UpdateQuery.rq)

```
PREFIX  :      <http://people.example.com/>
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>

SELECT ?mail ?name
WHERE {
  ?x  foaf:mbox   ?mail .
  OPTIONAL {
    ?x  foaf:name  ?name .
  }
}
```

Result

```
-----------------------------------------
| mail                          | name    |
=========================================
| <mailto:bob@example.net>      |         |
| <mailto:alice@example.net>    | "Alice" |
-----------------------------------------
```

SPARQL 1.1: Data Updates 3/4

Request (UpdateRequest.ru)

```
PREFIX  :      <http://people.example.com/>
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>

DELETE DATA {
  :bob  foaf:mbox  <mailto:bob@example.net> .
}
;
INSERT DATA {
  :bob    foaf:name   "Bob" ;
          foaf:mbox   <mailto:bob@bigcompany.com> .

  :fred   foaf:name   "Fred" ;
          foaf:mbox   <mailto:fred@example.net> ;
          foaf:knows  :alice .

  :alice  foaf:knows  :fred .
}
```

SPARQL 1.1: Data Updates 4/4

Query (UpdateQuery.rq)

```
PREFIX  :      <http://people.example.com/>
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>

SELECT ?mail ?name
WHERE {
  ?x  foaf:mbox   ?mail .
  OPTIONAL {
    ?x  foaf:name  ?name .
  }
}
```

Result

```
-----------------------------------------
| mail                          | name    |
=========================================
| <mailto:fred@example.net>     | "Fred"  |
| <mailto:bob@bigcompany.com>   | "Bob"   |
| <mailto:alice@example.net>    | "Alice" |
-----------------------------------------
```

Further Reading

- SPARQL 1.0

  http://www.w3.org/TR/rdf-sparql-query/

  http://jena.sourceforge.net/ARQ/Tutorial/
- SPARQL Protocol for RDF

  http://www.w3.org/TR/rdf-sparql-protocol/
- SPARQL 1.1 (Working Draft)

  http://www.w3.org/TR/sparql11-query/

  http://www.w3.org/TR/sparql11-property-paths/

  http://www.w3.org/TR/sparql11-update/