SPARQL1.1: An introduction

@AxelPolleres
Digital Enterprise Research Institute, National University of Ireland, Galway
What is SPARQL?

- **Query Language for RDF**
  - SQL “look-and-feel” for the Semantic Web
  - Means to query the Web of Data
  - Means to map between vocabularies
  - Means to access RDF stores

- **SPARQL1.0 (standard since 2008):**
  - Query Language
  - Protocol
  - Result Format

- **SPARQL1.1 (in progress):**
  - SPARQL 1.1 query language (additional features: aggregate functions, subqueries, negation, project expressions, property paths, basic federated queries)
  - SPARQL 1.1 Entailment regimes
  - SPARQL 1.1 Update: A full data manipulation language
  - SPARQL 1.1 Uniform HTTP Protocol for Managing RDF Graphs
  - SPARQL 1.1 Service Descriptions
What you’ll hear

- Run through SPARQL 1.0
- New features in SPARQL 1.1 Query
- SPARQL 1.1 Entailment Regimes
- Implementations, Status
RDF a plain data format for the Web

Various syntaxes, RDF/XML, Turtle, N3, RDFa,…

URIs, e.g.
- http://www.w3.org/2000/01/rdf-schema#label
- http://ontology.dumontierlab.com/isLocatedIn
- http://dbpedia.org/resource/Santiago
- http://dbpedia.org/resource/Chile

Blanknodes: “existential variables in the data” to express incomplete information, written as _:x or []

_L: x <http://www.w3.org/2000/01/rdf-schema#label> "Claudio Gutierrez" .

Literals, e.g.
- "2010"^^xsd:gYear
- "Brixen"@de
- "Bressanone"@it
- "Santiago"@es
- "Claudio Gutierres

"Santiago"@es

Chile

Santiago

Claudio Gutierrez

"Santiago"@es
RDF Data on the Web: Linked Open Data

The New York Times

July 2009
RDF Data online: Example 1/4

- (i) directly by the publishers
- (ii) by exporters

FOAF/RDF linked from a home page: personal data (foaf:name, foaf:phone, etc.), relationships foaf:knows, rdfs:seeAlso)
RDF Data online: Example 2/4

- (i) directly by the publishers
- (ii) by exporters, e.g. OpenLink’s Virtuoso.

E.g. DBPedia, an export of Wikipedia’s structured Data, using OpenLink’s Virtuoso (http://dbpedia.org)

Gives unique URIs to cities, countries, persons, etc. from wikipedia! E.g.,
http://dbpedia.org/resource/Santiago%2C_Chile
http://dbpedia.org/resource/Chile

Provides RDF version of all wikipedia structured data (infoboxes) and even a SPARQL query interface!
(i) directly by the publishers

(ii) by exporters, e.g. D2R.

e.g. L3S’ RDF export of the DBLP citation index, using FUB’s D2R (http://dblp.l3s.de/d2r/)

Gives unique URIs to authors, documents, etc. on DBLP! E.g.,
http://dblp.l3s.de/d2r/resource/authors/Tim_Berners-Lee,
http://dblp.l3s.de/d2r/resource/publications/journals/tplp/Berners-LeeCKSH08

Provides RDF version of all DBLP data and even a SPARQL query interface!
RDF Data online: Example 4/4

Tim Berners-Lee
Resource URI: http://dblp.l3s.de/d2r/resource/authors/Tim_Berners-Lee

Property | Value
--- | ---
is dc:creator of | <http://dblp.l3s.de/d2r/resource/publications/conf/aaai/KagalBCW06>
is dc:creator of | <http://dblp.l3s.de/d2r/resource/publications/conf/chi/schraefel/AWTBCJKDMSSW09>
is dc:creator of | <http://dblp.l3s.de/d2r/resource/publications/conf/esws/OmitolaKPYSSBGHsS10>
is dc:creator of | <http://dblp.l3s.de/d2r/resource/publications/conf/policy/HansonBKS07>
is dc:creator of | <http://dblp.l3s.de/d2r/resource/publications/conf/policy/KagalBCW06>

...  
foaf:homepage | <http://www.w3.org/People/Berners-Lee/>

rdfs:label | Tim Berners-Lee

is foaf:maker of | <http://dblp.l3s.de/d2r/resource/publications/conf/aaai/KagalBCW06>
is foaf:maker of | <http://dblp.l3s.de/d2r/resource/publications/conf/chi/schraefel/AWTBCJKDMSSW09>
is foaf:maker of | <http://dblp.l3s.de/d2r/resource/publications/conf/esws/OmitolaKPYSSBGHsS10>
is foaf:maker of | <http://dblp.l3s.de/d2r/resource/publications/conf/policy/HansonBKS07>
is foaf:maker of | <http://dblp.l3s.de/d2r/resource/publications/conf/policy/KagalBCW06>

...

foaf:name | Tim Berners-Lee

rdfs:seeAlso | <http://dblp.l3s.de/Authors/Authors+Tim+Berners-Lee>
rdfs:seeAlso | <http://www.bibsonomy.org/uri/author/Authors+Tim+Berners-Lee>
rdf:type | foaf:Agent
DBLP Data in RDF: Triples Turtle Syntax:

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .

<http://dblp.l3s.../journals/tplp/Berners-LeeCKSH08> rdf:type swrc:Article.
<http://dblp.l3s.../journals/tplp/Berners-LeeCKSH08> dcterms:issued "2008"^^xsd:gYear .
...
...
<http://dblp.l3s.../Tim_Berners-Lee> foaf:homepage <http://www.w3.org/People/Berners-Lee/> .
DBLP Data in RDF: Triples Turtle Syntax:

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .

<http://dblp.l3s.../journals/tplp/Berners-LeeCKSH08> rdf:type swrc:Article ;
  dcterms:issued "2008"^^xsd:gYear ;
  foaf:maker <http://dblp.l3s.../Tim_Berners-Lee> ,
              <http://dblp.l3s.../Dan_Connolly> ,
              <http://dblp.l3s.../Jim_Hendler> ,
              <http://dblp.l3s.../Lalana_Kagal> ,
              <http://dblp.l3s.../Yosi_Scharf> .

... 

<http://dblp.l3s.../conf/aaai/KagalBCW06> rdf:type swrc:inProceedings ;

... 

<http://dblp.l3s.../Tim_Berners-Lee> foaf:homepage <http://www.w3.org/People/Berners-Lee/> ;
  foaf:name "Tim Berners-Lee" .
```
Linked Data: What’s the point?

- Loads of **structured data** out there
- You want to do **structured queries** on top of it …
- SPARQL1.0 W3C Rec 15 January 2008… Now you can!
- Without exaggeration, SPARQL is probably a not too small a part of the LOD success story! … at least an important building block
How can I query that data? SPARQL

Basic graph pattern matching ~ Conjunctive queries

Example DBLP:

"Give me all documents by Tim Berners-Lee"

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?D
FROM <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>
WHERE {?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>}

FROM clause/Dataset can be implicit, e.g. when querying DBLP's SPARQL endpoint
SPARQL: Basic Graph Patterns

Basic graph pattern matching ~ Conjunctive queries

Example DBPEDIA:
“Give me all names of people born in Santiago”

Basic Graph Pattern (BGP) ... set of RDF triples with variables in S,P,O, e.g.:

{ ?P "born in" <http://dbpedia.org/resource/Santiago%2C_Chile>; "name" ?N }

How can I find the right properties for my query?
→ Look at the data!
SPARQL: Basic Graph Patterns

Basic graph pattern matching ~ Conjunctive queries

Example DBPEDIA:
“Give me all names of people born in Santiago”

Basic Graph Pattern (BGP) … set of RDF triples with variables in S,P,O, e.g.:

```
{ ?P dbpedia-owl:birthPlace <http://dbpedia.org/resource/Santiago%2C_Chile>;
  rdfs:label ?N }
```
SPARQL: Basic Graph Patterns

Basic graph pattern matching ~ Conjunctive queries

Example DBPEDIA:
“Give me all names of people born in Santiago”

PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?N
{ ?P dbpedia-owl:birthPlace <http://dbpedia.org/resource/Santiago%2C_Chile>;
  rdfs:label ?N }

Lesson learned: I can build SPARQL queries, from looking at the data and the URIs used (for properties and classes) in the data!
SPARQL: and how should I know all those prefixes? E.g. use prexif.cc !!!

PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?N
{?P dbpedia-owl:birthPlace <http://dbpedia.org/resource/Santiago%2C_Chile>;
  rdfs:label ?N }
SPARQL: Basic Graph Patterns

Basic graph pattern matching  ~  Conjunctive queries

Example DBLP:
“Give me all names of co-authors of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {
  _:D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>.
  _:D foaf:maker _:C.
  _:C foaf:name ?N
}.

• Blank nodes in Queries play a similar role as (non-distinguished) variables.
• Turtle style shortcuts are allowed (a bit extreme here, admittedly)
Avoid Duplicates: keyword **DISTINCT**

**Example DBLP:**

“Give me all names of co-authors of Tim Berners-Lee”

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?N
WHERE {
  [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
    [ foaf:name ?N ] ] . }
```

- **Blank nodes** in Queries play a *similar* role as (non-distinguished) variables.
- **Turtle style shortcuts** are allowed (*a bit extreme here, admittedly*)
SPARQL: Basic Graph Patterns

Basic graph pattern matching ~ Conjunctive queries

Example DBLP:

“Give me all names of co-authors of Tim Berners-Lee, their identifiers and their authored documents”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
  ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>.
  ?CoAuth foaf:name ?N
}

“SELECT *” outputs all variables in the pattern
More complex patterns in SPARQL 1.0

- UNION
- OPTIONAL
- FILTER
- Querying named GRAPHs
- Solution Modifiers (ordering, slicing/dicing results)
- ... plus some non-trivial combinations of these
UNIONs of conjunctive queries...

Unions of conjunctive queries

Example:
“Give me all names of co-authors or friends of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {
  \[ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
  foaf:name ?N \] .
  \}
  UNION
  {<http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
  \}

... of conjunctive queries...

Example:
“Give me all names of co-authors or friends of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {

}

Note: again Duplicates possible!

<table>
<thead>
<tr>
<th>?N</th>
<th>?N</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lalana Kagal&quot;</td>
<td>&quot;Michael Hausenblas&quot;</td>
<td>&quot;Jim Hendler&quot;</td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td>&quot;Jim Hendler&quot;</td>
<td>&quot;Charles McCathieNevile&quot;</td>
</tr>
<tr>
<td>&quot;Dan Connolly&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

U = "Lalana Kagal"
  "Tim Berners-Lee"
  "Dan Connolly"
  "Jim Hendler"
  "Michael Hausenblas"
  "Jim Hendler"
  "Charles McCathieNevile"
  ...

Note: again Duplicates possible!
Avoid Duplicates: keyword DISTINCT

Example:
“Give me all names of co-authors or friends of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?N
WHERE {
  { [ foaf:maker <http://dblp.l3s.de/…/authors/Tim_Berners-Lee>,
      [ foaf:name ?N ] ] . }
  UNION
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
    ?F foaf:name ?N }
}

=?N
"Lalana Kagal"
"Tim Berners-Lee"
"Dan Connolly"
"Jim Hendler"
...

=?N
"Michael Hausenblas"
"Jim Hendler"
"Charles McCathieNevile"
...

= [U]
UNIONS of conjunctive queries...

**Unions** of conjunctive queries

Example:

“Give me all names of co-authors or friends of Tim Berners-Lee”

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

SELECT ?CoAuthN ?FrN

WHERE {
  { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> [ foaf:name ?CoAuthN ] ] . }

UNION
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F . ?F foaf:name ?FrN }
}

<table>
<thead>
<tr>
<th>?CoAuthN</th>
<th>?FrN</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lalana Kagal&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Dan Connolly&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Michael Hausenblas&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Charles McCathieNevile&quot;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Optional parts in queries (Left Outer Join)

Example:

“Give me all names of co-authors of Tim Berners-Lee and optionally their homepage”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N ?H
WHERE {
  ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>.
  OPTIONAL { ?CoAuth foaf:homepage ?H }
}
FILTERING out query results

FILTERs allow to specify FILTER conditions on patterns

Example:

"Give me all names of co-authors of Tim Berners-Lee
and whose homepage starts with http://www.w3 different from Tim B.-L. himself"

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N ?H
WHERE {
  ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>.
  FILTER( regex( str(?H) , "^http://www.w3" ) &&
  ?CoAuth != <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> )
}
FILTERING out query results

**FILTERs** allow to specify FILTER conditions on pattern

- Can use an extensible library of built-in functions
  - Checking: `bound()`, `isIRI()`, `isBlank()`, `regex()` …
  - Conversion/extraction: `str()`, `datatype()`, `lang()` …
- Can be complex: `&&`, `||`, `!`
- **ATTENTION**: Evaluated in a 3-valued logic: `true`, `false`, `error`

**Example:**

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

SELECT ?N ?H
WHERE {
  ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>.
  OPTIONAL { ?CoAuth foaf:homepage ?H . }
  FILTER(! regex( str(?H) , "^http://www.w3" ) &&
    ?CoAuth != <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> )
}
```

Will result in **E** for unbound ?H

⇒ Whole FILTER expr always **E** for unbound ?H

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;m. c. schraefel&quot;</td>
<td><a href="http://www.ecs.soton.ac.uk/~mc/">http://www.ecs.soton.ac.uk/~mc/</a></td>
</tr>
<tr>
<td>&quot;Ryen White&quot;</td>
<td><a href="http://www.dcs.gla.ac.uk/~whiter/">http://www.dcs.gla.ac.uk/~whiter/</a></td>
</tr>
<tr>
<td>&quot;Desney S. Tan&quot;</td>
<td><a href="http://research.microsoft.com/%7Edesney/">http://research.microsoft.com/%7Edesney/</a></td>
</tr>
<tr>
<td>&quot;Jesse S. Kelano&quot;</td>
<td><a href="http://chin.org/~zeal/">http://chin.org/~zeal/</a></td>
</tr>
</tbody>
</table>
FILTERING out query results

- **ATTENTION**: FILTERs can NOT assign/create new values…

```sparql
PREFIX ex: <http://example.org/>
SELECT ?Item ?NewP
```

- Obviously, common query languages like SQL can do this…

```sql
SELECT Item, Price+10 AS NewPrice FROM Table
```

... FILTER in SPARQL is like WHERE in SQL, but SPARQL1.0 doesn’t have AS
Querying named GRAPHs

- **Find me people who have been involved with at least three ISWC or ESWC conference events.**
  (from SPARQL endpoint at data.semanticweb.org)

```sparql
SELECT ?person WHERE {
  GRAPH ?g1 { ?person a foaf:Person }
  GRAPH ?g2 { ?person a foaf:Person }
  GRAPH ?g3 { ?person a foaf:Person }
  FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) .
}
```

- The GRAPH ?g construct allows a pattern to match against one of the named graphs in the RDF dataset. The URI of the matching graph is bound to ?g (or whatever variable was actually used).
- The FILTER assures that we're finding a person who occurs in three *distinct* graphs.
Slicing and Dicing results

- **Solution Modifiers**
  - DISTINCT/REDUCED
  - ORDER BY
  - LIMIT/OFFSET

- **Example:**

```sql
SELECT DISTINCT ?person WHERE {
  GRAPH ?g1 { ?person a foaf:Person }
  GRAPH ?g2 { ?person a foaf:Person }
  GRAPH ?g3 { ?person a foaf:Person }
  FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) .

ORDER BY ?person
LIMIT 10
```
ASC, DESC, ORDER BY Expressions
More complex query examples 1/2

- **“IF-THEN-ELSE”**

  - “Give me the names of persons, if it exists, otherwise the nicknames, if it exists, otherwise the labels”

    ```sql
    SELECT ?X ?N
    WHERE{ ?X rdf:type foaf:Person
    OPTIONAL { ?X foaf:name ?N }
    OPTIONAL { ?X foaf:nickname ?N }
    OPTIONAL { ?X rdfs:label ?N } }
    ```

- **“Conditional OPTIONAL”**

  - “Give me the names and - only of those whose name starts with ‘D’ - the homepage”

    ```sql
    SELECT ?N ?H
    WHERE{ ?X foaf:name ?N
    OPTIONAL { ?X foaf:homepage ?H
    FILTER ( regex( str(?N), "^D" ) ) } }
    ```

- Non-compositionality raised some eyebrows... [Angles&Gutierrez, 2008] showed that compositional semantics can be achieved by rewriting.
Negation ("NOT EXISTS" in SQL)

- "Give me all Persons without a homepage"
- Option 1: by combination of OPTIONAL and FILTER(!bound(…) )

```sql
SELECT ?X
WHERE { ?X rdf:type foaf:Person
    OPTIONAL { ?X foaf:homepage ?H }
    FILTER( !bound( ?H ) ) }
```

- Option 2: by even weirder combination of OPTIONAL with GRAPH queries...

```sql
SELECT ?X
WHERE { ?X rdf:type foaf:Person
    OPTIONAL { ?X foaf:homepage ?H }
    GRAPH boundcheck.ttl {?H :is :unbound } }
```

Please forget this immediately again...

"These aren't the droids you're looking for"

where the aux. graph boundcheck.ttl contains the single triple [] :is :unbound.
Construct new graphs:

- “everybody knows their co-authors”

```
CONSTRUCT { ?X foaf:knows ?Y }
    FILTER ( ?X != ?Y ) }
```
Constructing Graphs

- Map between ontologies:
- E.g. for expressing complex ontology mappings between **FOAF** and **SIOC**
- “an sioc:name of a sioc:User is a foaf:nick”

Actually, expressible in new OWL2 (but not in OWL1):

```w3c-owl
foaf:nick owl:propertyChainAxiom (foaf:holdsAccount sioc:name)
```
Limitations

- Again, no assignment, creation of values
  - How to concatenate first name and last name?

- No aggregation (e.g. COUNT, SUM, ...):
  - How to create a graph that has publication count per person for DBLP?
  - No RDFS/OWL inference (so combining mappings in RDFS/OWL with queries in SPARQL not possible)
**SPARQL1.0 Formal Semantics**

- **Graph patterns:**
  - BGPs
  - $P_1 P_2$
  - $P \text{ FILTER } R$
  - $P_1 \text{ UNION } P_2$
  - $P_1 \text{ OPTIONAL } P_2$

- **Semantics**
  - $eval(D(G), \text{ graph pattern})$ … $D$ is a dataset,
    $G$ is the “active graph”
    recursively defined for all graph patterns in Section 12.5 of
    [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)

Spec. semantics is a bit hard to read …

Explained in more “accessible” terms in extended version of this
Tutorial: [http://www.polleres.net/presentations/20101006SPARQL1.1Tutorial.pptx](http://www.polleres.net/presentations/20101006SPARQL1.1Tutorial.pptx)
Academic works around SPARQL

- **SPARQL semantics**
  - [Perez et al. 2006] (pre-dates the spec) [Perez et al. 2009]

- **SPARQL equivalences**
  - also in [Perez et al. 2006],[Perez et al. 2009]
  - More in [Schmidt et al. 2010]

- **SPARQL expressivity**
  - Reducible to datalog with negation [Polleres 2007]
  - Other way around also works [Angles & Gutierrez 2008]

- **Proposed Extensions**
  - Aggregates [Polleres et al. 2007]
  - Property Paths [Alkhateeb et al. 2009], [Perez et al. 2008]
SPARQL 1.1

WG might still change some of the syntax/semantics definitions presented here based on community input.
This is where SPARQL1.1 starts

- Missing common feature requirements in existing implementations or requested urgently by the community:
  - Assignment/Project Expressions
  - Aggregate functions (SUM, AVG, MIN, MAX, COUNT, ...)
  - Subqueries
  - Property paths
    - complaint: SPARQL1.0 isn’t quite a “graph” query language

- Ease of use:
  - Why is Negation “hidden” in SPARQL1.0?

- Interplay with other SW standards:
  - SPARQL1.0 only defined for simple RDF entailment
  - Other Entailment regimes missing:
    - RDF(S), OWL
    - OWL2
    - RIF
Goals of SPARQL1.1

- **Per charter** ([http://www.w3.org/2009/05/sparql-phase-ll-charter.html](http://www.w3.org/2009/05/sparql-phase-ll-charter.html))
  - “The scope of this charter is to extend SPARQL technology to include some of the features that the community has identified as both desirable and important for interoperability based on experience with the initial version of the standard.”

  - No inclusion of new features that still require research
  - Upwards compatible with SPARQL1.0
  - The name SPARQL1.1 shall indicate an incremental change rather than any fundamental changes.
We will focus on these in today’s Tutorial

Goals of SPARQL 1.1

List of agreed features:

- **Additions to the Query Language:**
  - Project Expressions
  - Aggregate functions
  - Subqueries
  - Negation
  - Property Paths (*time permitting*)
  - Extend the function library (*time permitting*)
  - Basic federated Queries (*time permitting*)

- **Entailment** (*time permitting*)

- **SPARQL Update**
  - Full Update language
  - plus simple RESTful update methods for RDF graphs (HTTP methods)

- **Service Description**
  - Method for discovering a SPARQL endpoint’s capabilities
  - Summary of its data
Part 1: new query features

- Project Expressions
- Aggregate functions
- Subqueries
- Negation
- Property Paths
Assignments, Creating new values...

PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr * 1.1 AS ?NewP ) 
WHERE { ?Item ex:price ?Pr }

Data:

@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 .
ex:beer1 ex:price 3. 
ex:winel ex:price 3.50 .
ex:liqueurl ex:price "n/a".

Results:

<table>
<thead>
<tr>
<th>?Item</th>
<th>?NewP</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemonade1</td>
<td>3.3</td>
</tr>
<tr>
<td>beer1</td>
<td>3.3</td>
</tr>
<tr>
<td>winel</td>
<td>3.85</td>
</tr>
<tr>
<td>liqueurl</td>
<td></td>
</tr>
</tbody>
</table>

Leaves errors unbound!
Assignments, Creating new values...

PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr * 1.1 AS ?Pr )
WHERE { ?Item ex:price ?Pr }

Note: Variables “already bound” cannot be used for project expressions!
Aggregates
Aggregates

- “Count items”

PREFIX ex: <http://example.org/>
SELECT (Count(?Item) AS ?C)
WHERE { ?Item ex:price ?Pr }

Data:

@prefix ex: <http://example.org/> .

ex:lemonad1  ex:price 3 ;
    rdf:type ex:Softdrink.
ex:beer1      ex:price 3 ;
    rdf:type ex:Beer.
ex:wine1      ex:price 3.50 ;
    rdf:type ex:Wine.
ex:wine2      ex:price 4 .
    rdf:type ex:Wine.
ex:wine3      ex:price "n/a" ;
    rdf:type ex:Wine.

Results:

?C
5
Aggregates

- "Count categories"

PREFIX ex: <http://example.org/>
SELECT (Count(DISTINCT ?T) AS ?C)
WHERE { ?Item rdf:type ?T }

Data:

@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 ;
    rdf:type ex:Softdrink.
ex:beer1 ex:price 3 ;
    rdf:type ex:Beer.
ex:wine1 ex:price 3.50 ;
    rdf:type ex:Wine.
ex:wine2 ex:price 4 .
    rdf:type ex:Wine.
ex:wine3 ex:price "n/a" ;
    rdf:type ex:Wine.

Results:

?C
3
Aggregates - Grouping

“Count items per categories”

 PREFIX ex: <http://example.org/>
 SELECT ?T (Count(?Item) AS ?C)
 WHERE { ?Item rdf:type ?T }
 GROUP BY ?T

Data:
@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 ;
    rdf:type ex:Softdrink.
ex:beer1    ex:price 3 ;
      rdf:type ex:Beer.
ex:wine1    ex:price 3.50 ;
      rdf:type ex:Wine.
ex:wine2    ex:price 4 .
      rdf:type ex:Wine.
ex:wine3    ex:price "n/a" ;
      rdf:type ex:Wine.

Results:

<table>
<thead>
<tr>
<th>?T</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softdrink</td>
<td>1</td>
</tr>
<tr>
<td>Beer</td>
<td>1</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
</tr>
</tbody>
</table>
Aggregates – Filtering Groups

“Count items per categories, for those categories having more than one item”

PREFIX ex: <http://example.org/>
SELECT ?T (COUNT(?Item) AS ?C)
WHERE { ?Item rdf:type ?T }
GROUP BY ?T
HAVING COUNT(?Item) > 1

Data:

@prefix ex: <http://example.org/> .

<table>
<thead>
<tr>
<th>?T</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wine</td>
<td>3</td>
</tr>
</tbody>
</table>

ex:lemonade1 ex:price 3 ;
ex:beer1 ex:price 3 ;
ex:wine1 ex:price 3.50 ;
ex:wine2 ex:price 4 .
ex:wine3 ex:price "n/a" ;
Other Aggregates

- **SUM**  ... as usual
- **AVG**  ... as usual
- **MIN**  ... as usual
- **MAX**  ... as usual
- **SAMPLE**  ... “pick” one non-deterministically
- **GROUP_CONCAT**  ... concatenate values with a designated separator string

...this list is extensible  ... new built-ins will need to define error-behaviour, extra-parameters (like SEPARATOR in GROUP_CONCAT)
Example SUM

“Sum Prices per categories”

PREFIX ex: <http://example.org/>
SELECT ?T (Sum(IF(isNumeric(?Pr),?Pr,0) AS ?P)
WHERE { ?Item rdf:type ?T; ex:price ?Pr }
GROUP BY ?T

Data:
@prefix ex: <http://example.org/> .

ex:lemonade1 ex:price 3 ;
ex:beer1 ex:price 3 ;
ex:wine1 ex:price 3.50 ;
ex:wine2 ex:price 4 .
ex:wine3 ex:price "n/a";

Results:

<table>
<thead>
<tr>
<th>?T</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softdrink</td>
<td>3</td>
</tr>
<tr>
<td>Beer</td>
<td>3</td>
</tr>
<tr>
<td>Wine</td>
<td>7.5</td>
</tr>
</tbody>
</table>
Example GROUP_CONCAT, SAMPLE

- "pick one sample name per person, plus a concatenated list of nicknames"

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ( SAMPLE(?N) as ?Name) 
  ( GROUP_CONCAT(?M; SEPARATOR = "", ") AS ?Nicknames )
WHERE { ?P a foaf:Person ;
  foaf:name ?N ;
  foaf:nick ?M . }
GROUP BY ?P
```

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

ex:alice a foaf:Person; foaf:name "Alice Wonderland";
  foaf:nick "Alice", "The real Alice".

ex:bob a foaf:Person;
  foaf:name "Robert Doe", "Robert Charles Doe",
  "Robert C. Doe";
  foaf:nick "Bob", "Bobby", "RobC", "BobDoe".

ex:charles a foaf:Person;
  foaf:name "Charles Charles";
  foaf:nick "Charlie" .
Subqueries
Subqueries to realise complex mappings

- How to concatenate first name and last name?
- Now possible without problems per subqueries!

PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>

CONSTRUCT{ ?P foaf:name ?FullName }  
WHERE {
  WHERE { ?P foaf:firstName ?F ; foaf:lastName ?L. }  
}
Subqueries “Limit per resource”

- Give me all titles of papers of 10 persons who co-authored with Tim Berners-Lee

SELECT ?T
WHERE {
{
SELECT DISTINCT ?P
FILTER ( ?P != <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> )
}
LIMIT 10
}
}

- Returns titles for 10 persons, instead of just 10 rows
Attention: Subqueries do not allow to “inject values” from outside, but that limits some use cases, one might think of... e.g. an alternative “limit per resource” query:

```
SELECT ?P ?T
WHERE {
  ?P rdf:type Person .
  { SELECT ?T
    LIMIT 3 }
}
```

Different ?P/ different scope than the ?P outside of the subquery... i.e. no correlation:

<table>
<thead>
<tr>
<th>?P</th>
<th>?T</th>
</tr>
</thead>
<tbody>
<tr>
<td>:tim</td>
<td>“Doc3”</td>
</tr>
<tr>
<td>:tim</td>
<td>“Doc1”</td>
</tr>
<tr>
<td>:tim</td>
<td>“Doc2”</td>
</tr>
<tr>
<td>:tim</td>
<td>“Doc3”</td>
</tr>
</tbody>
</table>

... does NOT return 3 titles per author!
Note: At this point, no Dataset Clauses in Subselects, i.e.:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {
  { SELECT ?N
    FROM <http://www.w3.org/People/Berners-Lee/card>
    <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
    ?F foaf:name ?N
  }
  UNION
  { SELECT ?N
    FROM <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>
    { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
      [ foaf:name ?N ] ] . } }
MINUS and NOT EXISTS
MINUS and NOT EXISTS

- **Negation as failure in SPARQL1.0 is “ugly”:**

  SELECT ?X
  WHERE{ ?X rdf:type foaf:Person
     MINUS { ?X foaf:homepage ?H } }

- **SPARQL1.1 has two alternatives to do the same**
  - **NOT EXISTS in FILTERs**
    - detect non-existence
  - **(P1 MINUS P2 ) as a new binary operator**
    - “Remove rows with matching bindings”
    - only effective when P1 and P2 share variables
Property Path expressions

- Concatenate property paths, Arbitrary Length paths, etc.
- E.g. names of people Tim Berners-Lee transitivity co-authored papers with...

```
SELECT DISTINCT ?N
WHERE {
  <http://dblp.../Tim_Berners-Lee>,
  (^foaf:maker/foaf:maker)+/foaf:name ?N
}
```
Path expressions full list of operators

- **elt ... Path Element**

<table>
<thead>
<tr>
<th>Syntax Form</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>A URI or a prefixed name. A path of length one.</td>
</tr>
<tr>
<td><code>elt</code></td>
<td>Inverse path (object to subject).</td>
</tr>
<tr>
<td>`/uri or !(uri1</td>
<td>An expression that negates the property set. A URI which is not one of uri1, uri2, ...</td>
</tr>
<tr>
<td>`/uri and ![uri1</td>
<td>An expression that combines the property set with a negation. A URI which is not one of uri1, uri2, ...</td>
</tr>
<tr>
<td>(elt)</td>
<td>A group path elt, brackets control precedence.</td>
</tr>
<tr>
<td>elt1 / elt2</td>
<td>A sequence path of elt1, followed by elt2</td>
</tr>
<tr>
<td>elt1</td>
<td>/ elt2</td>
</tr>
<tr>
<td>elt*</td>
<td>A path of zero or more occurrences of elt.</td>
</tr>
<tr>
<td>elt+</td>
<td>A path of one or more occurrences of elt.</td>
</tr>
<tr>
<td>elt?</td>
<td>A path of zero or one elt.</td>
</tr>
<tr>
<td>elt{n,m}</td>
<td>A path between n and m occurrences of elt.</td>
</tr>
<tr>
<td>elt{n}</td>
<td>Exactly n occurrences of elt.</td>
</tr>
<tr>
<td>elt{n,}</td>
<td>n or more occurrences of elt.</td>
</tr>
<tr>
<td>elt{,n}</td>
<td>Between 0 and n occurrences of elt.</td>
</tr>
</tbody>
</table>

- **Semantics: by translation to native SPARQL with two core property paths**

  - ArbitraryPath(X, path, Y)
  - ZeroLengthPath(X, path, Y)
Path expressions

- Can be used for some ontological inference (well known since [Perez et al. 2008])
- E.g. Find all Beers in the Beer ontology

```
PREFIX beer: <http://www.purl.org/net/ontology/beer#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?beer
FROM <http://www.purl.org/net/ontology/beer>
WHERE {
}
```
Implementations of SPARQL 1.1 Query:

Some current (partial) SPARQL 1.1 implementations:

- **ARQ**
  - [http://sourceforge.net/projects/jena/](http://sourceforge.net/projects/jena/)
  - [http://sparql.org/sparql.html](http://sparql.org/sparql.html)

- **OpenAnzo**
  - [http://www.openanzo.org/](http://www.openanzo.org/)

- **Perl RDF**
  - [http://github.com/kasei/perlrdf/](http://github.com/kasei/perlrdf/)

- **Corese**

- **etc.**

Others probably forthcoming...

- **Loads of SPARQL 1.0 endpoints around**
  - Dbpedia: [http://dbpedia.org/snorql/](http://dbpedia.org/snorql/)
  - DBLP: [http://dblp.l3s.de/d2r/snorql/](http://dblp.l3s.de/d2r/snorql/)
  - Etc.
Part 2: Entailment Regimes

SPARQL 1.1 querying over RDFS+OWL2 ontologies and RIF rulesets?
SPARQL1.1 Entailment Regimes

- SPARQL1.1 will define SPARQL query answering over OWL2 ontologies and RIF rule sets:
  - [http://www.w3.org/TR/sparql11-entailment/](http://www.w3.org/TR/sparql11-entailment/)
  - RDF Entailment Regime
  - RDFS Entailment Regime
  - D-Entailment Regime
  - OWL 2 RDF-Based Semantics Entailment Regime
  - OWL 2 Direct Semantics Entailment Regime
  - RIF Core Entailment

  - Won’t go into details of those, but sketch the main ideas!
RDFS/OWL2 and SPARQL1.1

- General Idea: Answer Queries with implicit answers
- E.g. example from before:

```sql
PREFIX beer: <http://www.purl.org/net/ontology/beer#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?beer
FROM <http://www.purl.org/net/ontology/beer>
WHERE {
}
```

- beer:Boddingtons rdf:type beer:Ale .
- beer:Grafentrunk rdf:type beer:Bock .
- beer:Jever rdf:type beer:Pilsner .
- beer:Tetleys rdf:type beer:Ale .
Essential idea behind RDFS inference:

- SPARQL executes “inference” rules on the data, when answering queries, e.g.:


beer:Boddingtons rdf:type beer:Ale;
  rdf:type beer:TopFermentedBeer;
  rdf:type beer:Beer.
beer:Grafentrunk rdf:type beer:Bock.
  rdf:type beer:BottomFermentedBeer;
  rdf:type beer:Beer.
beer:Hoegaarden rdf:type beer:White;
  rdf:type beer:TopFermentedBeer;
  rdf:type beer:Beer.

...
General Idea: Answer Queries with implicit answers

E.g. Graph/Ontology:

```
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subclassOf
    [ a owl:Restriction ;
      owl:onProperty :hasFather ;
      owl:someValuesFrom foaf:Person ] .
foaf:knows rdfs:range foaf:Person.

:jeff a Person .
```

```
SELECT ?X { ?X a foaf:Person }

Pure SPARQL 1.0 returns only :Jeff, should also return :aidan
```
Challenges+Pitfalls:

- Possibly Infinite answers (by RDFS ContainerMembership properties, OWL datatype reasoning, etc.)
- Conjunctive Queries: non-distinguished variables
- SPARQL 1.1 features: Aggregates
Current Solution:

- Possibly Infinite answers (by RDFS ContainerMembership properties, OWL datatype reasoning, etc.)
  - Restrict answers to rdf:/rdfs:/owl:vocabulary minus rdf:_1 ... rdf:_n plus terms occurring in the data graph
- Non-distinguished variables
  - No non-distinguished variables, answers must result from BGP matching, projection a post-processing step not part of SPARQL entailment regimes.
- SPARQL 1.1 other features: e.g. Aggregates, etc.
  - Again not affected, answers must result from BGP matching, projection a post-processing step not part of entailment.

- Simple, BUT: maybe not always entirely intuitive, so
  - Good to know ;-)
Possibly Infinite answers RDF(S): Container Membership

Graph:
:rr2010Proceedings :hasEditors [ a rdf:Seq;
    rdf:_1 :pascal_hitzler;
    rdf:_2 :thomas_lukasiewicz
] .

Query with RDFS Entailment in mind:
SELECT ?CM { ?CM a rdfs:ContainerMembershipProperty}

Entailed by RDFS (axiomatic Triples):
  rdfs:_1 a rdfs:ContainerMembershipProperty .
  rdfs:_2 a rdfs:ContainerMembershipProperty .
  rdfs:_3 a rdfs:ContainerMembershipProperty .
  rdfs:_4 a rdfs:ContainerMembershipProperty .
...
Graph:
:rr2010Proceedings :hasEditors [ a rdf:Seq;
    rdf:_1 :pascal_hitzler;
    rdf:_2 :thomas_lukasiewicz
] .

Query with RDFS Entailment in mind:
SELECT ?CM { ?CM a rdfs:ContainerMembershipProperty}

SPARQL 1.1 restricts answers to rdf:/rdfs:/owl:vocabulary minus rdf:_1 ...
    rdf:_n plus terms occurring in the data graph

So, the only answers in SPARQL1.1 are:
{ ?CM/rdfs:_1, ?CM/rdfs:_2, }
Non-distinguished variables:

- **E.g. Graph**

```rml
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :hasFather ;
      owl:someValuesFrom foaf:Person ] .
foaf:knows rdfs:range foaf:Person.
:jeff a Person .

SELECT ?X ?Y { ?X :hasFather ?Y }
```

*No answer, because no known value for ?Y in the data graph.*
Non-distinguished variables:

- **E.g. Graph**

  ```
  foaf:Person rdfs:subClassOf foaf:Agent .
  foaf:Person rdfs:subClassOf
  [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person ] .
  foaf:knows rdfs:range foaf:Person.
  :jeff a Person .
  ```

  ```
  SELECT ?X { ?X :hasFather ?Y }
  ```

*But what about this one? ?Y looks like a “non-distinguished” variable*

*Solution: In SPARQL 1.1 answers must result from BGP matching, projection a post-processing step not part of entailment ➔ so, still no answer.*
Similar as before... aggregates are evaluated within algebra after BGP matching, so, no effect:

```
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person ] .
:jeff a Person .
foaf:knows rdfs:range foaf:Person.
```

```
SELECT ?X { ?X a foaf:Person }
```

Under RDFS/OWL entailment returns: {?X/jeff, ?X/aidan}
Similar as before... aggregates are evaluated as post-processing after BGP matching, so, no effect:

```sparql
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person ] .
:jeff a Person .
foaf:knows rdfs:range foaf:Person.
:jeff :hasFather [a Person].
:jeff owl:sameAs :aidan.
```

```
SELECT (Count(?X) AS ?Y) { ?X a foaf:Person }
```

Under RDFS/OWL entailment returns : {?Y/3}

Attention! owl:sameAs inference does NOT affect counting!!! ... But bnodes do!
RIF ... Rule Interchange format, Rec. since 2010

- RIF: Rule Interchange Format (rather than Rule language)
  - Framework for Rule Languages
  - Support RDF import: interesting for rule languages on top of RDF
  - Built-Ins support (close to XPath/XQuery functions and operators)
  - RIF Dialects:
    - RIF BLD: basic logic dialect = Horn rules with Built-ins, Equality
    - RIF Core: Datalog fragment (no logical function symbols, no head-equality)
    - RIF PRD: Production rules dialect
  - Normative XML syntax

- Commonalities with OWL:
  - RIF can model OWL2 RL
  - Share same Datatypes (XSD Datatypes, most OWL2 Datatypes)
  - Combinations of RIF with RDF, RDFS, and OWL defined in:
    http://www.w3.org/TR/rif-rdf-owl/
RIF Dialects

Core
- horn rules, monotonic
- datatypes & built-ins
- external functions
- Frames, class memberships
- equality (in conditions)
- ground lists
- existential quantification (in conditions)

BLD
- equality, class membership in conclusions
- frame subclasses
- open lists

PRD
- non-monotonic
- actions in conclusions
- negation
- subclasses
- membership in conclusion

SPARQL1.1 so far only defines Entailment for RIF Core... room for improvement (cf. e.g. Demo Obermeier et al. RR2010)
**SPARQL1.1 + RIF Core + RDFS/OWL**

- **RIF Core allows to encode RDFS, e.g.:**


- **RIF Core allows to encode OWL2 RL, e.g.:**

  owl1: { ?S1 owl:SameAs ?S2 } :-

- **Plus more (custom rules, including Built-ins):**


<http://ruleset1.rif>
How to reference to a RIF Ruleset from SPARQL?

- In OWL Entailment Regime, OWL is assumed to be part of the RDF Graph (OWL/RDF)

- RIF’s so far only a normative syntax is RIF/XML
  - RIF encoding in RDF (RIF/RDF) underway:
    [http://www.w3.org/2005/rules/wiki/RIF_In_RDF](http://www.w3.org/2005/rules/wiki/RIF_In_RDF)
  - Will also provide a new RDF property `rif:usedWithProfile` to import RIF rulesets (in RIF/XML or RIF/RDF). e.g.

```xml
<http://ruleset1.rif> rif:usedWithProfile
    <http://www.w3.org/ns/entailment/Simple> .
    <http://dblp.l3s.io/Tim_Berners-Lee>
        foaf:homepage <http://www.w3.org/People/Berners-Lee/> ;
        foaf:name "Tim Berners-Lee" .
    <http://www.w3.org/People/Berners-Lee/card#i>
        foaf:homepage <http://www.w3.org/People/Berners-Lee/> ;
        foaf:firstName "Timothy";
        foaf:lastName "Berners-Lee" .
```


<table>
<thead>
<tr>
<th>?P</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;dblp/Tim&gt;</code></td>
<td>Tim Berners-Lee</td>
</tr>
<tr>
<td><code>&lt;w3/B-Lee/card#i&gt;</code></td>
<td>Tim Berners-Lee</td>
</tr>
<tr>
<td><code>&lt;dblp/Tim&gt;</code></td>
<td>Timothy Berners-Lee</td>
</tr>
<tr>
<td><code>&lt;w3/B-Lee/card#i&gt;</code></td>
<td>Timothy Berners-Lee</td>
</tr>
</tbody>
</table>
Wrapping up

- **SPARQL 1.0**
  - UNIONs of Conjunctive Queries, FILTERs, GRAPH queries, OPTIONAL, (hidden) negation
  - contributed largely to the current Linked Data boom
  - Inspired interesting academic work

- **SPARQL 1.1**
  - A reasonable next step
    - Incorporating highly demanded features
    - Closing the gaps to neighbour standards (OWL2, RIF)
  - Not all of it is trivial → SPARQL1.1 takes a very pragmatic path

- Hopefully inspiring for more research, more data, and more applications!
List of agreed features:

- **Additions to the Query Language:**
  - Project Expressions
  - Aggregate functions
  - Subqueries
  - Negation
  - Property Paths *(time permitting)*
  - Extend the function library *(time permitting)*
  - Basic federated Queries *(time permitting)*

- **Entailment** *(time permitting)*

- **SPARQL Update**
  - Full Update language
  - plus simple RESTful update methods for RDF graphs (HTTP methods)

- **Service Description**
  - Method for discovering a SPARQL endpoint’s capabilities
  - Summary of its data
Extended Function Library

Functions Library in SPARQL1.0 is insufficient:

- Bound(. )
- isLiteral(. )
- isBlank(. )
- isIRI(. )
- Str(. )
- Regex(. , .)
- +,-,*, <, >, =

New functions to be included in standard library:

- COALESCE, IF
- Functions from the Xpath/Xquery function library
  - String manipulation, more math, etc. ... e.g. fn:concat

Essentially: rubber-stamp common functions present in current implementations
Basic federated Queries (*time permitting*)

- [http://www.w3.org/TR/sparql11-federated-query/](http://www.w3.org/TR/sparql11-federated-query/)
  - Will be integrated in Query spec
- **Essentially new pattern** SERVICE
  - Similar to GRAPH
  - allows delegate query parts to a specific (remote) endpoint

Recall: *We were cheating in this query before!!*

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {
  {<http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
   ?F foaf:name ?N }
  UNION
  { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
    [ foaf:name ?N ] ] . }
}
```
Basic federated Queries (time permitting)

- [http://www.w3.org/TR/sparql11-federated-query/](http://www.w3.org/TR/sparql11-federated-query/)
  - Will be integrated in Query spec
- Essentially new pattern SERVICE
  - Similar to GRAPH
  - allows delegate query parts to a specific (remote) endpoint

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
    ?F foaf:name ?N } 
  UNION 
  { SERVICE <http://dblp.l3s.de/d2r/sparql>
    { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
        [ foaf:name ?N ] ] . } } 
}
```
Sometimes you want to “inject” or “fix” some bindings into the query to be sent to an external endpoint.

Goal: reduce data to be transferred:

Example:

```sql
... WHERE { ?s :p2 ?v2 } BINDINGS ?s ?v2 { ( <s1> 7 ) ( <s2> UNBOUND ) }
```

```sql
... WHERE { { ?s :p2 ?v2 }
    { SELECT ( <s1> AS ?s ) ( 7 AS ?v2 ) WHERE {} } 
    UNION 
    { SELECT ( <s2> AS ?s ) WHERE {} } }
```

→ i.e. can be viewed as “syntactic sugar”, may be helpful...
Like SQL ... SPARQL/RDF Stores need a standard Data Manipulation Language
http://www.w3.org/TR/sparql11-update/

SPARQL 1.1 Update Language

- Graph Update
  - INSERT DATA
  - DELETE DATA
  - DELETE/INSERT
  - DELETE
  - INSERT
  - DELETE WHERE
  - LOAD
  - CLEAR

- Graph Management
  - CREATE
  - DROP

Issue: Graph-aware stores vs. Quad Stores
Base vocabulary to describe

- **features of SPARQL endpoints**
- **datasets** (via vocabularies external to the Spec, e.g. VOID)

http://www.w3.org/TR/sparql11-service-description/

### 3.2 Classes
- 3.2.1 `sd:Service`
- 3.2.2 `sd:Language`
- 3.2.3 `sd:Function`
- 3.2.4 `sd:Aggregate`
- 3.2.5 `sd:EntailmentRegime`
- 3.2.6 `sd:EntailmentProfile`
- 3.2.7 `sd:GraphCollection`
- 3.2.8 `sd:Dataset`
- 3.2.9 `sd:Graph`
- 3.2.10 `sd:NamedGraph`

### 3.3 Instances
- 3.3.1 `sd:SPARQL10Query`
- 3.3.2 `sd:SPARQL11Query`
- 3.3.3 `sd:SPARQL11Update`
- 3.3.4 `sd:DereferencesURIs`
- 3.3.5 `sd:UnionDefaultGraph`
- 3.3.6 `sd:RequiresDataset`
- 3.3.7 `sd:EmptyGraphs`

### 3.4 Properties
- 3.4.1 `sd:url`
- 3.4.2 `sd:feature`
- 3.4.3 `sd:defaultEntailmentRegime`
- 3.4.4 `sd:supportedEntailmentProfile`
- 3.4.5 `sd:entailmentRegime`
- 3.4.6 `sd:extensionFunction`
- 3.4.7 `sd:extensionAggregate`
- 3.4.8 `sd:languageExtension`
- 3.4.9 `sd:supportedLanguage`
- 3.4.10 `sd:propertyFeature`
- 3.4.11 `sd:defaultDatasetDescription`
- 3.4.12 `sd:availableGraphDescriptions`
- 3.4.13 `sd:resultFormat`
- 3.4.14 `sd:defaultGraph`
- 3.4.15 `sd:namedGraph`
- 3.4.16 `sd:name`
- 3.4.17 `sd:graph`
Relevant W3C Specs

- SPARQL Query Language for RDF [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
- SPARQL1.1 Query Language for RDF (working draft) [http://www.w3.org/TR/sparql11-query/](http://www.w3.org/TR/sparql11-query/)
- SPARQL1.1 Entailment Regimes (working draft) [http://www.w3.org/TR/sparql11-entailment/](http://www.w3.org/TR/sparql11-entailment/)

RDF(S) Entailment/D-Entailment:
- RDF Semantics [http://www.w3.org/TR/rdf-mt/](http://www.w3.org/TR/rdf-mt/)

OWL Entailment:
- OWL2 Web Ontology Language Primer [http://www.w3.org/TR/owl2-primer/](http://www.w3.org/TR/owl2-primer/)
- OWL2 Web Ontology Language Profiles [http://www.w3.org/TR/owl2-profiles/](http://www.w3.org/TR/owl2-profiles/)

RIF Entailment:
- RIF Core Dialect [http://www.w3.org/TR/rif-core/](http://www.w3.org/TR/rif-core/)
- RIF Basic Logic Dialect [http://www.w3.org/TR/rif-bld/](http://www.w3.org/TR/rif-bld/)
- RIF RDF and OWL compatibility [http://www.w3.org/TR/rif-rdf-owl/](http://www.w3.org/TR/rif-rdf-owl/)
References: Academic Results on SPARQL


Acknowledgements

- The members of the W3C SPARQL WG, particularly Lee Feigenbaum, who I stole some examples from
- The members of the W3C RIF WG

- The EU project **NET²** which sponsored my visit.